Applications of Surface Networks to Sampling Problems in Computer Graphics

Thesis by

Brian Von Herzen

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1989

(Submitted July 20, 1988)

Caltech-CS-TR-88-15

Acknowledgments

My thanks go to Al Barr for his advice on this dissertation and on long-term planning. I thank my committee, Al Barr, Joel Burdick, Fred Culick, Jim Kajiya, and Carver Mead, for their careful review of this dissertation. I thank my classmates Ronen Barzel, Tim Kay, John Platt, Dave Kirk, Mike Newton, Bill Dally, Jed Lengyel, John Snyder, Bill Athas, Dave Wark, and Jim Flowers, from whom I learned a lot.

Wen-King Su made it a lot easier to do the figures in this thesis, and Scott Hemphill and Glenn Tesler developed the TEX previewer. I thank Dian De Sha for proofreading, and Carolyn Collins for administrative assistance.

Thanks go to Lane for suggestions on English style and usage. Thanks go to Dick for the global perspective. Thanks go to Jan for making education a top priority. And thanks to Rebecca Truman, my graduate experience was longer and more enjoyable.

The research described in this thesis was sponsored in part by the Fannie and John Hertz Foundation, Schlumberger Palo Alto Research Center, International Business Machines, Inc., Hewlett-Packard Co., and Apple Computer, Inc.

Abstract

This thesis develops the theory, algorithms and data structures for adaptive sampling of parametric functions, which can represent the shapes and motions of physical objects. For the first time, ensured methods are derived for determining collisions and other interactions for a broad class of parametric functions. A new data structure, called a *surface network*, is developed for the collision algorithm and for other sampling problems in computer graphics. A surface network organizes a set of parametric samples into a hierarchy. Surface networks are shown to be good for rendering images, for approximating surfaces, and for modeling physical environments. The basic notion of a surface network is generalized to higher-dimensional problems such as collision detection. We may think of a two-dimensional network covering a three-dimensional solid, or an n-dimensional network embedded in a higher-dimensional space. Surface networks are applied to the problems of adaptive sampling of static parametric surfaces, to adaptive sampling of time-dependent parametric surfaces, and to a variety of applications in computer graphics, robotics, and aviation.

First we develop the theory for adaptive sampling of static surfaces. We explore bounding volumes that enclose static surfaces, subdivision mechanisms that adjust the sampling density, and subdivision criteria that determine where samples should be placed.

A new method is developed for creating bounding ellipsoids of parametric surfaces using a Lipschitz condition to place bounds on the derivatives of parametric functions. The bounding volumes are arranged in a hierarchy based on the hierarchy of the surface network. The method ensures that the bounding volume hierarchy contains the parametric surface completely. The bounding volumes are useful for computing surface intersections. They are potentially useful for ray tracing of parametric surfaces.

We develop and examine a variety of subdivision mechanisms to control the sampling process for parametric functions. Some of the methods are shown to improve the robustness of adaptive sampling. Algorithms for one mechanism, using bintrees of right parametric triangles, are particularly simple and robust.

A set of empirical subdivision criteria determine where to sample a surface, when we have no additional information about the surface. Parametric samples are concentrated in regions of high curvature, and along intersection boundaries.

Once the foundations of adaptive sampling for static surfaces are described, we examine time-dependent surfaces. Based on results with the empirical subdivision criteria for static surfaces, we derive ensured criteria for collision determination. We develop a new set of rectangular bounding volumes, apply a standard

k-dimensional subdivision mechanism called k-d trees, and develop criteria for ensuring that we detect collisions between parametric surfaces.

We produce rectangular bounding boxes using a "Jacobian"-style matrix of Lipschitz conditions on the parametric function. The rectangular method produces even tighter bounds on the surface than the ellipsoidal method, and is effective for computing collisions between parametric surfaces.

A new collision determination technique is developed that can detect collisions of parametric functions, based on surface network hierarchies. The technique guarantees that the first collision is found, to within the temporal accuracy of the computation, for surfaces with bounded parametric derivatives. Alternatively, it is possible to guarantee that no collisions occur for the same class of surfaces. When a collision is found, the technique reports the location and parameters of the collision as well as the time of first collision.

Finally, we examine several applications of the sampling methods. Surface networks are applied to the problem of converting a two-dimensional image, or texture map, into a set of triangles that tile the plane. Many polygon-rendering systems do not provide the capability of rendering surfaces with textures. The technique converts textures to triangles that can be rendered directly by a polygon system. In addition, potential applications of the collision determination techniques are discussed, including robotics and air-traffic control problems.

Contents

[1]published in *Computer Graphics 21,* 4, 1987, pp. 103-112.

Iconic Index of Figures

Nomenclature

backface culling: The process of removing parts of a surface that face away from the viewer. In renderings that eliminate hidden surfaces, we can see only those parts of objects that face the viewer. The rendering system overhead is reduced by eliminating backfacing polygons before they are drawn.

bicubic patch: A parametric surface defined by cubic equations of two parameters, u and v. The isoparametric curves of a bicubic patch are always cubic. The general form of the equation is

$$\begin{aligned}\vec{P}(u,v) = {} & \vec{A}_{11}u^3v^3 + \vec{A}_{12}u^3v^2 + \vec{A}_{13}u^3v + \vec{A}_{14}u^3 \\ & + \vec{A}_{21}u^2v^3 + \vec{A}_{22}u^2v^2 + \vec{A}_{23}u^2v + \vec{A}_{24}u^2 \\ & + \vec{A}_{31}uv^3 + \vec{A}_{32}uv^2 + \vec{A}_{33}uv + \vec{A}_{34}u \\ & + \vec{A}_{41}v^3 + \vec{A}_{42}v^2 + \vec{A}_{43}v + \vec{A}_{44}.\end{aligned}$$

bintree: A tree with a branching ratio of two. Each node in a bintree may have up to two subnodes.

biquadratic patch: A parametric surface defined by quadratic equations of two parameters, u and v. See *bicubic patch.*

biquartic patch: A parametric surface defined by quartic equations of two parameters, u and v. See *bicubic patch.*

bivariate function: A function of two variables, such as a parametric surface.

Boolean boundary: An intersection curve of two surface manifolds; an intersection boundary.

Boolean subdivision criterion: A criterion that controls the subdivision process near Boolean intersection boundaries. For instance, subdivision occurs where the intersection boundary of two surfaces is curved in screen space.

bounding box inequality: A set of inequalities defining the interior region of a bounding box. For Cartesian coordinates in three dimensions, we have

$$\begin{aligned} |x - x_c| &\leq \Delta x \\ |y - y_c| &\leq \Delta y \\ |z - z_c| &\leq \Delta z, \end{aligned} \tag{0.1}$$

where (x_c, y_c, z_c) is the point at the center of the box, and $(\Delta x, \Delta y, \Delta z)$ are the bounding box radii.

bounding box radii: The half-widths of a bounding box along each of the coordinate axes.

bounding volume: A volume that completely encloses an object. Algorithms based on bounding volumes can confirm that two objects do not intersect. If the bounding volumes do not intersect, then neither do the objects.

C_0 continuity (of a function): Continuity of the zeroth derivative of a function or, in other words, continuity of the function itself.

C_1 continuity (of a function): Continuity of the first derivative of a function.

chopping (triangles): Triangles are clipped by the inside-outside function of intersecting parametric surfaces. Triangle chopping reduces a fringe that may occur along the intersection boundary of two surfaces (See Figure 7.A).

collision: An event where two objects come into contact. See *ϵ-collision.*

culling: The process of removing surface elements from consideration that are not visible for some reason. For example, backface culling removes polygons facing away from the viewer, and surface intersection culling removes polygons clipped by other surfaces.

curvature subdivision criterion: A subdivision criterion based on the local curvature of the parametric surface.

deformation: A modeling operation that bends, twists, tapers, or otherwise alters the shape of a parametric surface.

disjoint: Sets are *disjoint* if they have no elements in common. Point sets are disjoint if they have no points in common. Parametric surfaces are disjoint if they do not intersect.

ϵ-collision: A state where the minimum separation between two objects is less than or equal to a distance tolerance ϵ.

fixed-point representation: A representation for numbers that has a fixed number of bits of precision for the integer part and for the fractional part of a number.

Gouraud shading: Interpolative shading across a triangle where the color is specified at each vertex, and is linearly interpolated across the interior of the triangle to shade the entire triangle. The interpolation is done first along the edges of the triangle, and then along scanlines that traverse the interior of the triangle.

hashed array: An array whose entries have been evenly redistributed by a hashing function. This operation is a useful technique in searching problems, where an element in a large array must be found. For example, the samples in a surface quadtree are not evenly distributed in u and v. Hashing on the u and v parameters redistributes the samples so that they evenly populate an array, facilitating storage and accessing operations.

hashing function: A function that maps a record of information onto an array index in a pseudorandom fashion, in an effort to evenly redistribute the elements across the array.

inside-outside function: A function indicating whether a point is inside or outside a solid. The function returns a negative value if the coordinate is inside the solid, a positive value if the point is outside the solid, and a value of zero if the point is on the surface of the solid. Linear root-finding techniques frequently converge faster if the inside-outside function is roughly linear with distance to the surface.

intersection boundary: The intersection curve of two surface manifolds.

isoparametric contour: A line of constant parametric value, for instance ($u = 0.5$).

isothetic rectangle: A rectangle whose sides are parallel to the coordinate axes.

Jacobian matrix: An $n \times m$ matrix of the partial derivatives of an n-vector parametric function of m variables. Each row of the matrix is used for a separate component of the output vector; and each column of the matrix has a separate parametric variable. For the function $\vec{f}(u,v,t) = (x,y,z)^T$, the

Jacobian J is defined by

$$\mathbf{J}(u,v,t) \equiv \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} & \frac{\partial z}{\partial t} \end{pmatrix}.$$

k-d tree: A k-dimensional binary search tree. Each node in a k-d tree represents a box in k dimensions. At each transition from one level to the next, the box is split in half along one of the coordinate axes. In this manner, it is possible to divide a k-dimensional box into smaller boxes using a binary tree. See Figure 4.4 for an example of a two-dimensional tree.

L-functions: An L-function is a parametric function that satisfies a Lipschitz condition. An L-function consists of a parametric function definition, a parametric domain over which the definition applies, and a Lipschitz value L that bounds the parametric derivatives. For example, we construct an L-function for a parabola:

$$\left\{ f(x) = x^2, \quad x \in [x_0, x_1], \quad L = 2\max(|x_0|, |x_1|) \right\}. \tag{0.2}$$

The value of L is the maximum slope of the function $f(x)$ over the specified domain $x \in [x_0, x_1]$.

Lipschitz condition: A parametric function $\vec{f}(\vec{u})$ satisfies a Lipschitz condition if and only if

$$\left\| \vec{f}(\vec{u}_2) - \vec{f}(\vec{u}_1) \right\| \leq L \left\| \vec{u}_2 - \vec{u}_1 \right\|,$$

for some finite value of the Lipschitz constant L ([Gear 71]).

Lipschitz constant: A value L that satisfies a Lipschitz condition for a function $\vec{f}(\vec{u})$. The Lipschitz constant places an upper bound on the rate of change of the function $\vec{f}(\vec{u})$ over a parametric region R.

mach bands: Bright or dark lines caused by artifacts of the human visual system when viewing a continuous intensity function whose slope is discontinuous. Mach bands may be produced at the edges of adjacent, smoothly shaded polygons.

manhattan distance: A distance metric, which is defined in two dimensions by

$$D((u_1, v_1), (u_2, v_2)) = |u_1 - u_2| + |v_1 - v_2|,$$

for points (u_1, v_1) and (u_2, v_2).

mip map: A technique for texture mapping that represents an image at multiple resolutions. The coarse images have been low-pass filtered to reduce aliasing, while the fine images have all the spatial frequency components. Samples are taken from one of the maps, depending on how much filtering is needed for the particular application. This technique avoids the filtering computation at sampling time.

modeling hierarchy: A tree of modeling operations, such as rotations, translations, scalings, bends, twists, tapers, intersections; and primitives, such as quadric surfaces, bicubic splines, half-planes, and profile solids. The operations are combined in a hierarchy that specifies the scope of each operation, creating a model of a three-dimensional solid.

modeling space: A three-dimensional coordinate system centered around the model being rendered. The curvature subdivision criterion measures curvature in modeling space to control the sampling process.

parametric sample: A set consisting of a parametric vector $(u_1, u_2, \ldots, u_n)$ and the value of a function $\vec{f}(u_1, u_2, \ldots, u_n)$ for that parametric vector.

parametric sampling: The process of evaluating a function for a finite set of parametric values.

parametric space: A two-dimensional coordinate system spanning the domain of a parametric surface. The coordinates (u, v) are usually used to represent a point in parametric space. The domain of u and v is usually the interval $[0, 1]$.

parametric space wrap-around: Some closed surfaces, such as superquadrics and supertori, are periodic in u and/or v. Cracks may appear in the surface between the edges $u = 0$ and $u = 1$ if care is not taken to match the sample points along each edge.

parametric surface: A vector function $\vec{P}(u, v)$ of two scalar variables that describes a surface embedded in three-dimensional space, or a surface manifold. The default range of u and v is the interval $[0, 1]$, for the parametric functions described in this work.

pixel: A picture element; an abstraction of a local intensity on a regular grid of intensities.

planar subdivision A collection of closed line segments that partition the plane and intersect only at segment endpoints. Planar subdivisions are useful for eliminating cracks in polygonal approximations of parametric surfaces.

principal directions (of curvature of a surface): A pair of local coordinate axes tangent to a parametric surface that point in the directions of maximum and minimum curvature. These axes always lie perpendicular to each other.

profile solid: The interior region bounded by a profile surface.

profile surface: A surface formed from two curves: a vertical (or north-south) profile, and a horizontal (or east-west) profile.

proximity subdivision criterion: A criterion that forces subdivision where surfaces potentially intersect. The criterion explores local minima of the absolute value of the inside-outside function.

quadric surface: A surface defined by an algebraic, second-degree equation. The quadric surfaces are the ellipsoids, hyperboloids, and paraboloids.

quadtree: A tree data structure with a branching ratio of four. Each node in a quadtree may have up to four subnodes. In image processing and computer graphics, a quadtree frequently also refers to a square region that is subdivided recursively into smaller squares.

rate matrix: Given a continuous parametric function $\vec{f}(u,v,t)$, a rate matrix is a matrix $\mathbf{M}$ that satisfies the condition

$$\begin{aligned} |x(u,v,t) - x(u_c,v_c,t_c)| &\leq M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ |y(u,v,t) - y(u_c,v_c,t_c)| &\leq M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ |z(u,v,t) - z(u_c,v_c,t_c)| &\leq M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t, \end{aligned}$$

over a rectangular region R, where

$$R = \left\{ (u,v,t)^T : \ |u-u_c| \leq \Delta u, \ \ |v-v_c| \leq \Delta v, \ \ |t-t_c| \leq \Delta t \ \right\}. \quad (0.3)$$

in the domain of $\vec{f}(u,v,t)$. A sufficient value of $\mathbf{M}$ for differentiable functions is given by

$$M_{ij} \geq \max_R |J_{ij}|,$$

where $\mathbf{J}$ is the Jacobian matrix of the function $\vec{f}(u,v,t)$.

rational polynomial surface: A parametric surface whose coordinates are given by the ratio of two polynomial functions in u and v.

ray tracing: A rendering method based on tracing rays of light backwards from the viewpoint to the objects in the scene. Rays are refracted and reflected off the objects to produce realistic highlights and shadows not possible with many other techniques.

recursive subdivision: The process of repeatedly subdividing a region until a set of subdivision criteria has been satisfied.

regula falsi **iteration:** An iterative procedure for finding a root of a function, given a positive value and a negative value for the function. The root of the function is estimated to be where the line joining the positive sample value and the negative sample value passes through zero. If this new sample value is positive, it replaces the old positive sample; if it is negative, then it replaces the negative sample. Iteration continues until the value obtained is sufficiently close to zero.

rendering (of surfaces): The process of creating an image of a surface.

restricted quadtree: A special type of quadtree whose neighboring squares may differ in size by at most a factor of two.

right triangular subdivision: A triangular subdivision consisting only of right triangles.

root (of a quadtree): The first element of a quadtree, which consists of a single square to be recursively subdivided into smaller squares.

scanline: A horizontal row of pixels in a digital image that extend from the left border of the image to the right border. Many rendering algorithms compute one scanline at a time.

screen space: A coordinate system centered on the image being generated by the rendering system. Several subdivision criteria use screen space measurements to determine the visible "badness" of a rendering artifact.

silhouette subdivision criterion: A criterion governing subdivision of regions that cross the silhouette boundary of a surface.

silhouette: The border of a surface as seen from a particular viewpoint.

square (of a quadtree): An element of a quadtree. Every element in the quadtree hierarchy is a square.

subdivision criterion: A decision-making algorithm that determines the parametric locations on a surface that require additional sampling.

subdivision mechanism: An algorithm or production rule that determines the process for obtaining additional samples in a parametric region.

summed-area table: A table T representing an image I that stores at each location the sum of the intensities of all the pixels that lie in the upper-left quadrant relative to that location:

$$T_{mn} \equiv \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}.$$

This representation permits the fast integration of intensities within any isothetic rectangle in the image, and is useful for filtering texture maps.

superellipse: A curve defined parametrically by $x = \cos^p(\theta)$, and $y = \sin^p(\theta)$, where θ is the parameter, and p is a constant.

superquadric: A profile surface made from two superellipses.

surface intersection biasing: A technique of offsetting the inside-outside function of a surface so that the polygons of the two surfaces do not touch. This procedure reduces artifacts associated with finite precision arithmetic.

surface manifold: A continuous, piecewise differentiable surface that can be embedded in three-dimensional or higher-dimensional space.

surface network: A network of parametric surface samples arranged in a simple hierarchy.

surface quadtree: A quadtree that spans the parameter space of a surface. Additional samples are created by subdividing squares into smaller squares.

tetrahedral subdivision: A volumetric subdivision whose interior regions consist only of tetrahedra.

texture map: An image that represents the surface coloration or texture of an object. The texture is effectively mapped onto the surface. The boundaries of the map usually correspond to isoparametric contours.

triangular subdivision: A planar subdivision whose interior regions consist only of triangles.

uniform subdivision criterion: A criterion that forces parametrically uniform sampling of a surface.

unit square: A square defined by the interval $[0, 1]$ in two coordinate directions. The domain of the parametric surfaces used in this paper is bounded by the unit square in parameter space.

volumetric subdivision: A collection of closed triangles that partition three-dimensional space, and that intersect only at triangle edges.

Chapter 1

Introduction

1.1 Motivation

Parametric functions are widely used in computer graphics for the modeling of complex three-dimensional objects, and for modeling their motions as a function of time ([Kajiya and Snyder 88], [Barr 86], [Sederberg and Parry 86]). It is intuitively simple to generate and manipulate parametric functions. All that is necessary is to sample the function across its domain, and generate surface elements from the samples. The goal is to obtain a good approximation to the shape of a surface, using a finite number of parametric samples.

The basic method is to evaluate the function for a finite set of parametric values, and somehow to interpolate the surface between the known points. This method is referred to as *parametric sampling.*

The simplest approach to representing parametric functions is to cover the domain of the function with a parametrically uniform grid of samples, and to generate surface elements from adjacent samples. Unfortunately, some parametric regions may need to be sampled much more finely than others. Using this technique, either one region is undersampled or another is oversampled, resulting in computational inefficiency.

An alternative approach is to concentrate samples where they are needed the most. Several issues arise when considering adaptive techniques: What criteria should determine the local concentration of samples? What data structures are effective at storing samples at variable densities? How do we ensure that we have an adequate sampling of the surface? And what applications benefit from an adaptive approach? This thesis will address these questions for static and for time-dependent parametric functions.

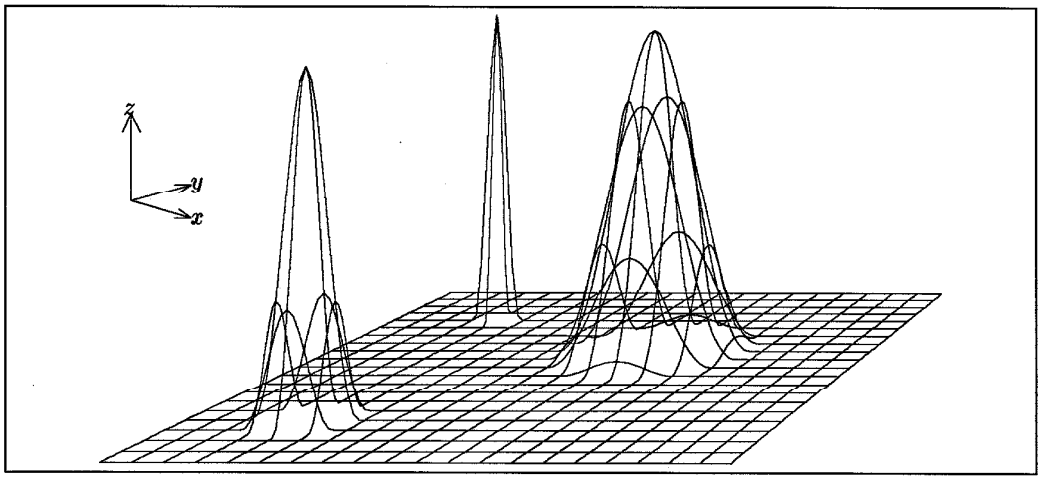

Figure 1.1: Parametric spike functions can be made arbitrarily sharp, such that their detection is extremely difficult. We need some other information in order to guarantee the detection of the spike.

1.2 Surface Networks

The standard uniform sampling techniques require only an array to store parametric samples. Adaptive sampling techniques require more flexible data structures, ones that can represent the surface hierarchically at a variety of resolutions. It is necessary to adjust the density of samples across a surface, as needed, for various sampling criteria. A new data structure, called a *surface network*, hierarchically stores surface samples from parametric functions. Algorithms based on surface networks can generate surface elements, determine intersections between parametric functions, and provide a recursive mechanism for subdivision of surface elements. Although it may be easiest to think of surface networks in terms of two-dimensional manifolds, the basic data structures extend to parametric functions of n dimensions, including time. This means that we can adaptively sample the position of an object as a function of time.

1.3 Bounding Volumes for Parametric Functions

There is a fundamental problem with the use of parametric functions in computer graphics: If you don't know the accuracy of a surface approximation, how can you be sure that the approximation is any good at all? Most of the standard techniques in computer graphics ignore the question of accuracy of surface approximations

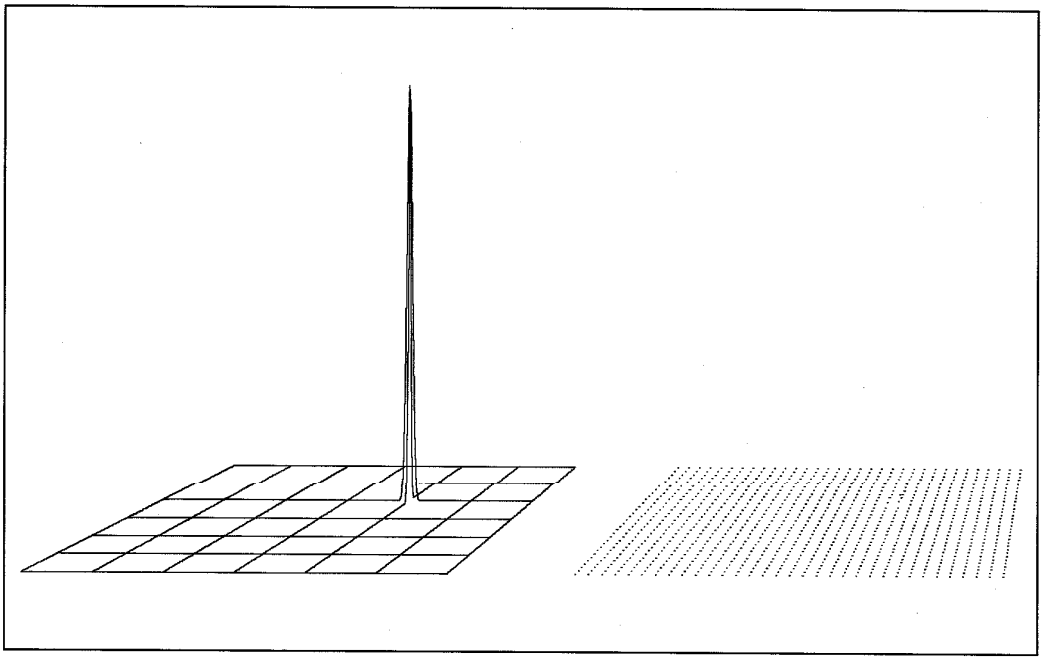

Figure 1.2: The left illustration shows a sharp parametric spike function. The right illustration shows the information provided to the sampling algorithm, given only the parametric samples, which are drawn as dots. The spike function falls between the samples.

([Foley and Van Dam 82]). The standard method is to sample uniformly across a surface, look at an image of the result, and increase the uniform sampling density if the image is not satisfactory. As we increase the complexity of geometric models in computer animations, we increase the probability of gross sampling errors with standard methods.

As an example, Figure 1.1 illustrates a set of parametric spike functions of various widths. Each spike is a parametric Gaussian bump with a different horizontal scaling. As the parametric width of the spike is reduced, so is the chance of finding the spike. Figure 1.2 shows a sharp spike that falls between the samples of the parametric function. It is not sufficient to have a finite set of samples from the parametric surface. To be sure of finding the spike, we need additional information.

This thesis develops the theory for guaranteeing, with upper bounds on parametric derivatives, that surface approximations are accurate to within an arbitrary, finite tolerance. We consider certain types of parametric functions, which we refer to as *L-functions*, that satisfy a Lipschitz condition ([Gear 71]). The Lipschitz condition places an upper bound on the parametric rate of change of the function. An

L-function consists of a parametric function definition, a parametric domain over which the definition applies, and a Lipschitz value L that bounds the parametric derivatives. For example, we construct an L-function for a parabola:

$$\left\{ f(x) = x^2, \quad x \in [x_0, x_1], \quad L = 2\max(|x_0|, |x_1|) \right\}. \tag{1.1}$$

The value of L is the maximum slope of the function $f(x)$ over the specified domain $x \in [x_0, x_1]$. The only requirements on an L-function are that it be continuous and have a finite rate of change. We must be able to determine a bound on the rate of change.

1.4 Original Results

The following original results are developed in this thesis:

- A new data structure is developed for adaptive sampling of parametric surfaces, called a *surface network*. A surface network is a hierarchical structure of parametric samples distributed across a surface. Surface networks are good for rendering, for surface approximations, and for computer modeling. The basic notion of a surface network may be generalized to higher-dimensional problems such as collision detection.

- A new method for creating bounding ellipsoids of parametric surfaces and sub-regions is developed using a Lipschitz condition on the parametric functions. The Lipschitz condition places a constraint on the parametric derivatives of the surface.

- A new method is developed for creating rectangular bounding boxes of parametric surfaces using the Jacobian matrix of the surface. This method uses a type of Lipschitz condition, called the *rate condition*, and produces even tighter bounds on the surface than the ellipsoidal method.

- A new collision determination technique is developed that can detect collisions of parametric functions. The technique guarantees that the first collision is found, to within the temporal accuracy of the computation, for surfaces with a rate condition. Alternatively, it is possible to guarantee that no collisions occur for the same class of surfaces. Potential applications to robotics and to air traffic control are discussed.

- Surface networks are applied to the problem of converting a two-dimensional image, or texture map, into a set of triangles that tile the plane. Many

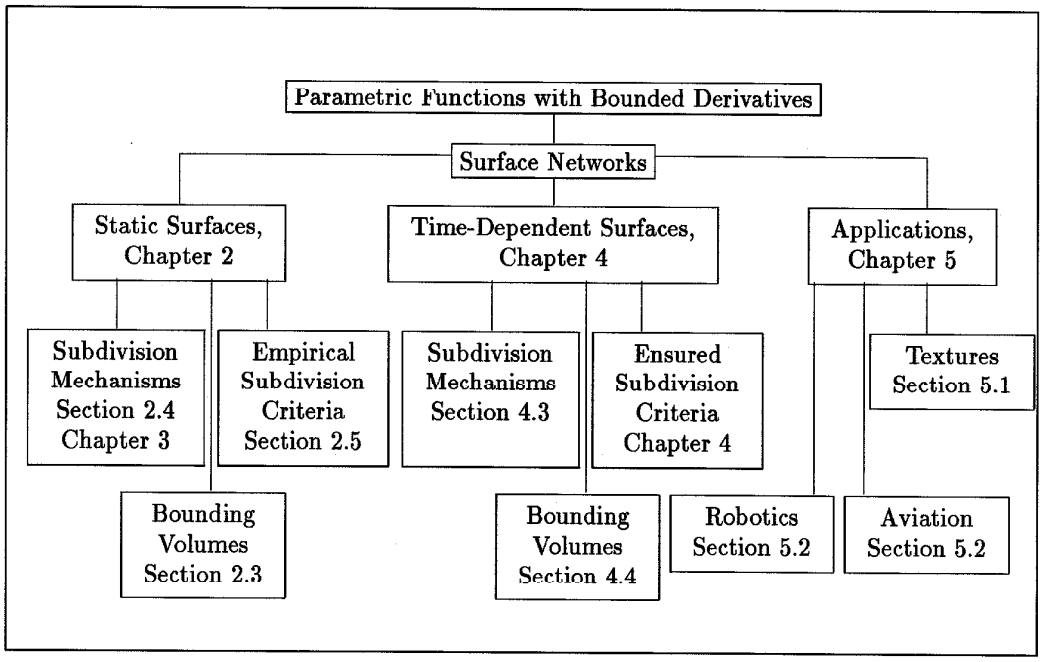

Figure 1.3: Breakdown of the adaptive sampling problems examined in this thesis for parametric surfaces.

polygon-rendering systems do not provide the capability of rendering surfaces with textures. The technique converts textures to triangles that can be rendered directly by a polygon system.

- A series of subdivision mechanisms are developed to control the sampling process for parametric functions. These mechanisms improve the robustness of adaptive sampling. Algorithms for one mechanism, which uses bintrees of right parametric triangles, are relatively simple and robust.

1.5 Summary

The headings in Figure 1.3 show the decomposition of the adaptive sampling problem into distinct subproblems. We examine issues related to the accurate representation of static parametric functions, which apply to geometric modeling in computer graphics. The results for static surfaces are extended to surfaces that move as a function of time. The results for time-dependent surfaces apply to collision problems in dynamical simulations. Adaptive sampling techniques are

also applied to texture maps for surface coloration of objects, and to anticollision systems in robotics and in aviation.

Chapter 2 develops the notion of applying surface networks to the problem of accurately representing static parametric surfaces. Accuracy is defined in terms of the distance from a parametric function to its approximation that uses finite numbers of surface elements. We develop the basic theory for applying the Lipschitz condition to the problem of creating a set of bounding volumes that are guaranteed to enclose the parametric surface completely. The bounding volumes can guarantee the accuracy of the surface network. A subdivision mechanism, called *restricted quadtrees*, is shown to be superior to unrestricted quadtrees at adaptive sampling of parametric surfaces. A set of empirical subdivision criteria are developed for parametric surfaces without Lipschitz conditions; these produce reasonable surface approximations in cases where we do not have any information about a parametric function other than a finite set of samples.

In Chapter 3, we take a detailed look at alternative subdivision mechanisms. A progression of quadtrees and bintrees are examined and compared. One technique, using bintrees of right triangles, requires fewer samples than restricted quadtrees, yet retains many of its advantages, including the smooth variation in sampling frequency across the parametric surface. We generalize the method to bintrees of non-right triangles, and extend the results to tetrahedral subdivisions. The surface elements generated from these subdivision mechanisms may be planar, such as triangles, or non-planar, such as biquadratic or Steiner patches ([Sederberg and Anderson 86]).

In Chapter 4, we develop the theory for parametric functions to guarantee the determination of collisions between surfaces for functions with a Lipschitz condition. The theory is first developed for interference between static surfaces. A rectangular subdivision mechanism called k-d trees is used, which readily extends to k parametric dimensions. We extend the results to parametric surfaces that move as a function of time. The algorithm finds all *ϵ-collisions*, where the minimum separation between two surfaces is less than a distance tolerance, ϵ. We are able to guarantee that we will find an ϵ-collision , if there is one, and that we can find the first ϵ-collision to within the temporal accuracy of the computation.

Chapter 5 discusses applications of surface networks to the problem of generating a set of polygons that approximate two-dimensional images, or texture. We use bintrees of right triangles as a subdivision mechanism, and develop subdivision criteria for various types of images. Examples are illustrated using images taken from playing cards. The polygonal tilings are used for an animation of dynamic constraints applied to a house of cards. We also explore potential applications of the collision determination theory to the problems of robotic path verification and collision prediction for aircraft.

Chapter 2

Accurate Sampling of Deformed, Intersecting Surfaces†

2.1 Overview

In this chapter a quadtree algorithm is developed to triangulate deformed, intersecting parametric surfaces. The biggest problem with adaptive sampling is guaranteeing that the triangulation is accurate within a given tolerance. A new method guarantees the accuracy of the triangulation, given a "Lipschitz" condition on the surface definition. The method constructs a hierarchical set of bounding volumes for the surface, useful for ray tracing and solid modeling operations. The task of adaptively sampling a surface is broken into two parts: a subdivision mechanism for recursively sampling a surface, and a set of subdivision criteria for determining where to sample the surface.

An adaptive sampling technique is said to be robust if it accurately represents the surface being sampled. A new type of quadtree, called a *restricted quadtree,* is more robust at adaptive sampling of parametric surfaces than is the traditional unrestricted quadtree. Each subregion in the quadtree is half the width of the previous region. The restricted quadtree requires that adjacent regions be the same width within a factor of two, while the traditional quadtree makes no restriction on neighbor width. Restricted surface quadtrees are effective at recursively sampling a parametric surface. Quadtree samples are concentrated in regions of high curvature and along intersection boundaries, using several subdivision criteria. Silhouette subdivision improves the accuracy of the silhouette boundary when a viewing transformation is available at sampling time. The adaptive sampling method is more robust than uniform sampling, and can be more efficient at

†published in *Computer Graphics 21,* 4, 1987, pp. 103-112.

rendering deformed, intersecting parametric surfaces.

2.2 Introduction

2.2.1 Motivation for Studying Surface Sampling Techniques

Interesting mathematical formulations for deformed surfaces have been recently developed ([Barr 84], [Sederberg and Parry 86]). Robust and efficient algorithms for rendering these surfaces, however, do not exist currently. We need to render highly curved regions on deformed surfaces correctly. In addition, we must render such critical areas as silhouette boundaries and surface intersections accurately.

The simplest sampling algorithm covers a surface with a parametrically uniform grid of samples, and divides the surface into small polygons that are easy to render. Frequently, aliasing occurs in highly curved regions, because of the constant sampling rate. Regions of low curvature are oversampled, thus wasting polygons.

An alternative approach is to concentrate samples where they are needed the most. A quadtree subdivision technique is presented here that concentrates samples in regions of high curvature and in other critical regions. The technique is more robust than uniform sampling of parametric surfaces, and is more efficient for the examples given here.

With uniform sampling of parametric surfaces, a tradeoff must be made between image quality and number of polygons. Adaptive sampling can produce better images with fewer polygons than can the uniform sampling approach. The uniform chain in Figure 2.1 shows an image that took 53 minutes on an IBM 4341 using uniform sampling; most of that time was spent on small regions of the image. The nearest link is not sampled adequately, as compared with the quadtree image of the same object that took 45 minutes to compute, as shown in Figure 2.2. The CPU times represent the time required to sample complex bending and twisting deformation functions of parametric primitives for nearly 100,000 sample points.

2.2.2 Background

Quadtree algorithms have been developed for applications in screen space and in parametric space. Quadtrees have been used to rasterize polygons ([Warnock 69]). Quadtree subdivision in screen space has been used as an antialiasing technique in ray tracing ([Whitted 80]). Bicubic patches have been rendered using recursive subdivision techniques in the parametric space of the surface ([Catmull 75],

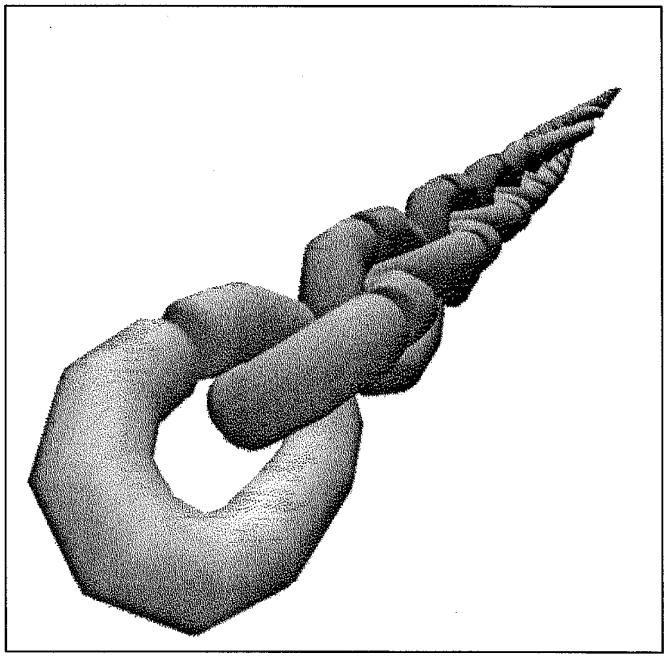

Figure 2.1: Uniformly sampled chain, 60,000 polygons, sampling time 53 minutes.

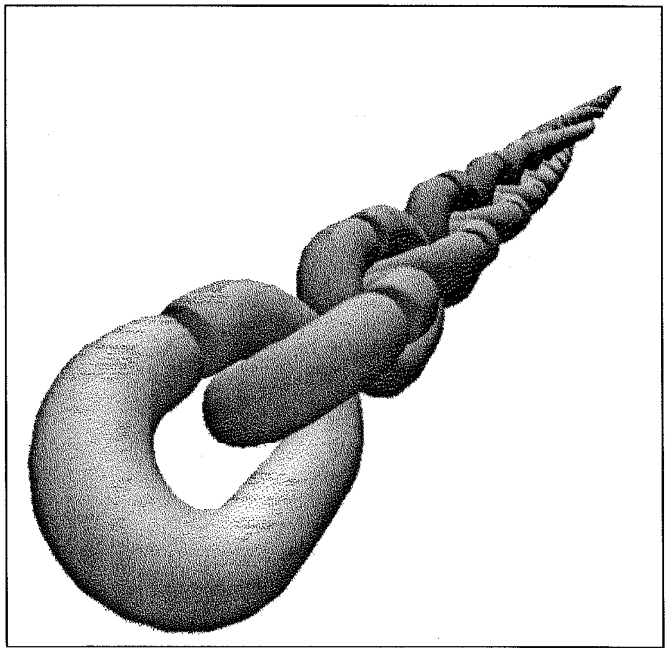

Figure 2.2: Quadtree chain, 50,000 polygons, sampling time 45 minutes.

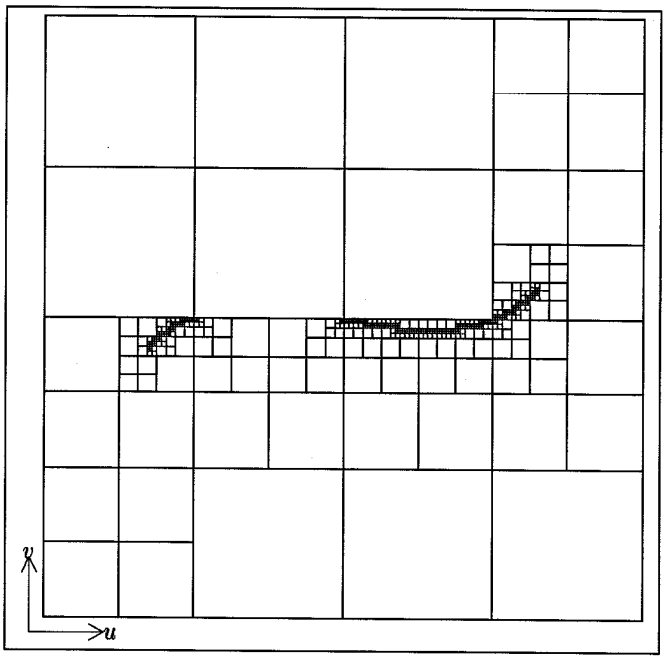

Figure 2.3: An unrestricted quadtree omits the local maximum in a cubic curve.

[Carlson 82], [Schweitzer and Cobb 82]). Methods have been developed for the display and intersection of piecewise polynomial surfaces ([Lane and Riesenfeld 80]). Recursive techniques have been described for sampling parametric surfaces using a curvature subdivision criterion ([Lane and Carpenter 79]). Scan-line techniques have been used to render a variety of parametric surfaces ([Blinn 78]). Adaptive subdivision has been used to fit surfaces to sampled data ([Schmitt, Barsky, Du 86]).

When ray-tracing complex surfaces, it is often attractive to break them into triangles before rendering ([Barr 86], [Kay and Kajiya 86], [Snyder and Barr 87]). The memory required for such a decomposition varies directly with the number of triangles. Adaptive subdivision greatly reduces the number of triangles needed to describe a surface accurately. In addition, a new application of the Lipschitz condition to parametric surfaces can guarantee the accuracy of polygonal approximations.

A new type of quadtree, the restricted quadtree, is described here for subdividing a surface. Restricted quadtrees differ from unrestricted quadtrees in that neighboring squares may differ in width by at most a factor of two. While the unrestricted quadtree does not reliably follow such features as the local maximum of the curve shown in Figure 2.3, the restricted quadtree is much more robust at adequately sampling curves and surfaces (Figure 2.4). An efficient set of subdivision

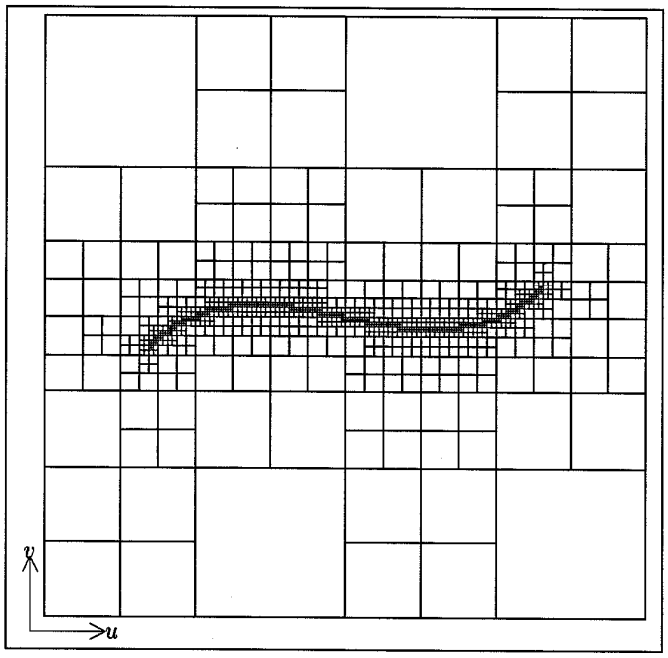

Figure 2.4: The restricted quadtree samples the cubic curve adequately.

criteria controls the sampling process for rendering deformed, intersecting surfaces. Several subdivision criteria are examined, including curvature subdivision, intersection subdivision, and potential intersection subdivision. A view-dependent subdivision criterion causes subdivision at the silhouette boundary from a given viewpoint. Each region is subdivided if any of the subdivision criteria are not met. Regions of low curvature are represented with a few relatively large polygons, while regions of high curvature are represented with many little polygons.

2.3 Bounding Volumes for Parametric Surfaces

Given a continuous parametric surface, we will show how to construct bounding volumes that contain the surface. Bounding volumes are useful for intersection operations, for collision detection, for ray tracing of parametric surfaces, and for accurately triangulating a parametric surface.

The ideal method guarantees that each part of the surface remains within its bounding volume. The task is impossible, however, if we are given a completely arbitrary parametric surface and no additional information. We know what the function values are at every surface point we have sampled. If the samples are the

only information available to us, there is no way to guarantee what the function is doing at places we have not sampled. However, with a single additional value, called a Lipschitz constant, we can bound the surface.

2.3.1 The Lipschitz Condition for a Parametric Curve

First, we define the Lipschitz condition for a parametric curve $\vec{f}(u)$ over a domain $0 \leq u \leq 1$ (Figure 2.5) ([Lin and Segel 74]). The Lipschitz condition states that

$$|\vec{f}(u_1) - \vec{f}(u_0)| \leq L|u_1 - u_0|, \tag{2.1}$$

where L is the Lipschitz constant, and u_0 and u_1 are in the interval $0 \leq u_i \leq 1$. The Lipschitz constant bounds the maximum parametric derivative of the function $\vec{f}(u)$, namely, $|d\vec{f}/du| \leq L$. Consider a value u_2 between $u = 0$ and $u = 1$. We have

$$|\vec{f}(u_2) - \vec{f}(0)| \leq L|u_2 - 0| \text{ and} \tag{2.2}$$
$$|\vec{f}(1) - \vec{f}(u_2)| \leq L|1 - u_2|. \tag{2.3}$$

Adding the equations gives us

$$|\vec{f}(u_2) - \vec{f}(0)| + |\vec{f}(1) - \vec{f}(u_2)| \leq L(u_2 - 0) + L(1 - u_2) = L. \tag{2.4}$$

We take the limiting case:

$$d_1 + d_2 = L, \text{ with} \tag{2.5}$$
$$d_1 = |\vec{f}(u_2) - \vec{f}(0)| \text{ and} \tag{2.6}$$
$$d_2 = |\vec{f}(1) - \vec{f}(u_2)|. \tag{2.7}$$

The solution of the equation is an ellipsoid with foci $\vec{f}(0)$ and $\vec{f}(1)$ such that the sum of the distances from the foci to any point on the ellipsoid is equal to L. Basically, we have a string of fixed length that is tied at both ends to the points $\vec{f}(0)$ and $\vec{f}(1)$.

2.3.2 The Lipschitz Condition for a Parametric Surface

Now we extend the result to parametric surfaces of two variables, $\vec{f}(u, v)$. The Lipschitz condition on a parametric surface $\vec{f}(\vec{P})$ over a domain $\vec{P} = (u, v)$ is defined by:

$$|\vec{f}(\vec{P}_1) - \vec{f}(\vec{P}_0)| \leq L|\vec{P}_1 - \vec{P}_0|, \tag{2.8}$$

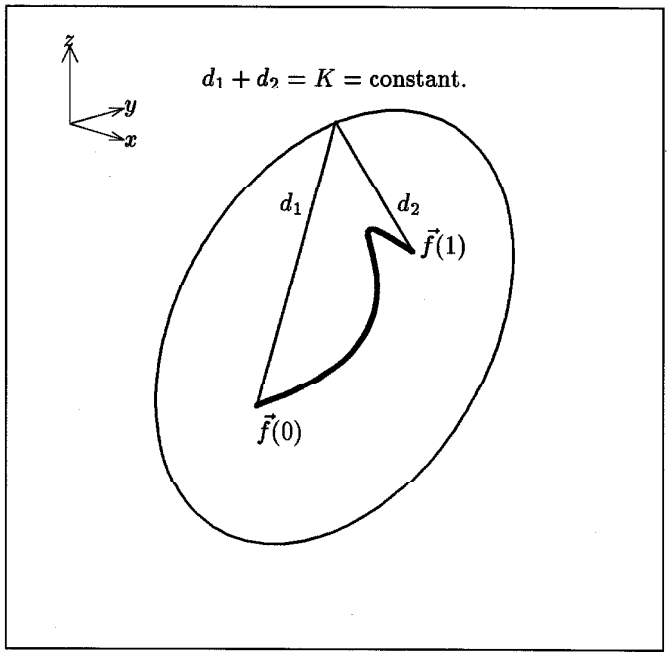

Figure 2.5: Bounding ellipsoid for a parametric curve $\vec{f}(u), 0 \leq u \leq 1$. The length of the curve must be less than or equal to L.

where $\vec{P_0}$ and $\vec{P_1}$ are any two points in the parametric space of the surface, and L is the Lipschitz constant. The constant L determines a maximum distance between two points in modeling space, given the distance of the two points in parameter space. We use the L_2 norm for the Lipschitz equation: $\|\Delta\vec{V}\|_2 = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$,

$$\|\vec{f}(\vec{P_1}) - \vec{f}(\vec{P_0})\|_2 \leq L\|\vec{P_1} - \vec{P_0}\|_2. \tag{2.9}$$

The L_2 norm represents the Euclidean distance between any two points. In contrast, the L_1 norm represents the sum of perpendicular distances between any two points. The L_1 norm is given by $\|\Delta V\|_1 = |\Delta x| + |\Delta y| + |\Delta z|$, and we always have $\|\Delta\vec{V}\|_2 \leq \|\Delta\vec{V}\|_1$, since the shortest distance between any two points falls along a straight line. Therefore, we can use the L_2 norm in modeling space and the L_1 norm in parametric space:

$$\|\vec{f}(\vec{P_1}) - \vec{f}(\vec{P_0})\|_2 \leq L\|\vec{P_1} - \vec{P_0}\|_2 \leq L\|\vec{P_1} - \vec{P_0}\|_1. \tag{2.10}$$

The L_1 norm is attractive because the distance from $\vec{P_0}$ to $\vec{P_1}$ via $\vec{P}$ is independent of $\vec{P}$ (Figure 2.6).

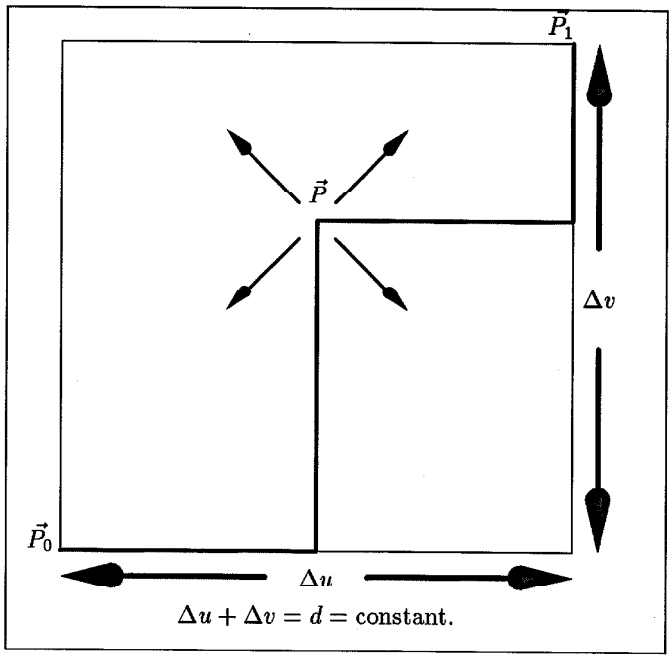

Figure 2.6: L_1 distance from $\vec{P}_0$ to $\vec{P}_1$ via $\vec{P}$. The L_1 distance d is independent of $\vec{P}$.

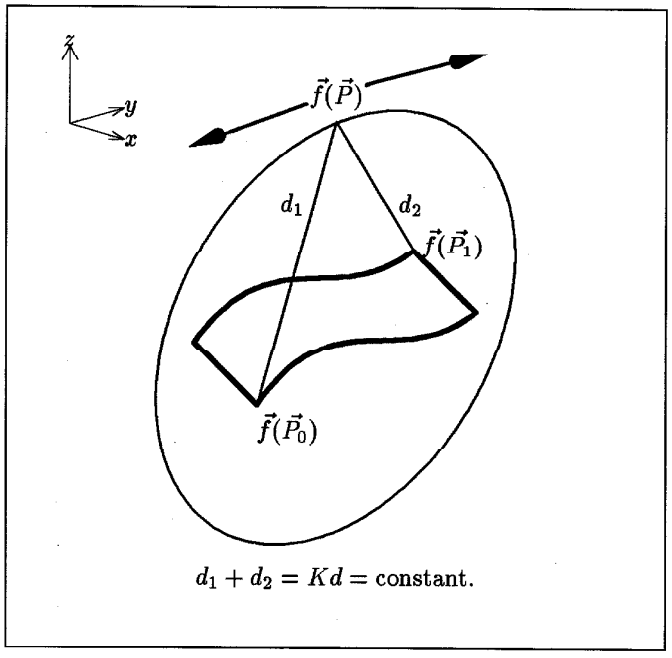

Figure 2.7: Bounding ellipsoid for the parametric surface $\vec{f}(\vec{P})$.

We evaluate the Lipschitz condition from $\vec{P}_0$ to $\vec{P}$ and from $\vec{P}$ to $\vec{P}_1$, where $\vec{P}$ is anywhere in the rectangle formed by opposite corners $\vec{P}_0$ and $\vec{P}_1$:

$$\|\vec{f}(\vec{P}) - \vec{f}(\vec{P}_0)\|_2 \leq L\|\vec{P} - \vec{P}_0\|_1, \text{ and} \tag{2.11}$$
$$\|\vec{f}(\vec{P}_1) - \vec{f}(\vec{P})\|_2 \leq L\|\vec{P}_1 - \vec{P}\|_1. \tag{2.12}$$

Adding the two equations,

$$\|\vec{f}(\vec{P}) - \vec{f}(\vec{P}_0)\|_2 + \|\vec{f}(\vec{P}_1) - \vec{f}(\vec{P})\|_2 \leq L\|\vec{P} - \vec{P}_0\|_1 + L\|\vec{P}_1 - \vec{P}\|_1. \tag{2.13}$$

Taking the limiting case,

$$d_1 + d_2 - L((u - u_0) + (v - v_0)) + L((u_1 - u) + (v_1 - v)) = C, \tag{2.14}$$
$$d_1 + d_2 = C, \text{ where} \tag{2.15}$$
$$d_1 = \|\vec{f}(\vec{P}) - \vec{f}(\vec{P}_0)\|_2, \tag{2.16}$$
$$d_2 = \|\vec{f}(\vec{P}_1) - \vec{f}(\vec{P})\|_2, \text{ and} \tag{2.17}$$
$$C = L((u_1 - u_0) + (v_1 - v_0)). \tag{2.18}$$

The solution of the equation is an ellipsoid with foci $\vec{f}(\vec{P}_0)$ and $\vec{f}(\vec{P}_1)$ such that the sum of the distances from the foci to any point on the ellipsoid is equal to C. Again, we have a string of fixed length, tied at both ends to the points $\vec{f}(\vec{P}_0)$ and $\vec{f}(\vec{P}_1)$ (Figure 2.7).

We now have a mechanism for enclosing any region or subregion of the parametric surface in a bounding ellipsoid. Given a surface quadtree, every region of the quadtree has a bounding ellipsoid that encloses it. The same hierarchy used in the quadtree may be used to construct a hierarchy of bounding volumes ([Kay and Kajiya 86]). The hierarchy of bounding volumes is useful for ray tracing, for collision detection, and for guaranteeing the accuracy of a triangulation. Each region in a surface quadtree has a maximum error based on its bounding ellipsoid. Given a desired error tolerance, we can subdivide the regions until the tolerance is reached. We obtain an arbitrarily accurate sampling of the surface.

2.3.3 Determining the Lipschitz Constant

The Lipschitz constant may be derived directly from the parametric equation by taking the global maximum of the parametric derivatives of the surface. We assume that the parametric domain of the surface spans the region $R : 0 \leq u \leq 1$, $0 \leq v \leq 1$. If the surface is differentiable, we can use the parametric derivatives to

obtain the Lipschitz constant L:

$$L \geq \max_R \left(\left\| \frac{\partial \vec{f}(u,v)}{\partial u} \right\|_1, \left\| \frac{\partial \vec{f}(u,v)}{\partial v} \right\|_1 \right). \tag{2.19}$$

This value of L can be shown to be necessary and sufficient to satisfy Eqn. 2.11. The condition is necessary from the case where Δv is set to zero, and we solve for L. And the equation is sufficient for other cases, from the triangle inequality.

The technique is feasible for surfaces such as bicubics, superquadrics, and low-order polynomials, because local maxima are easy to find in the parametric derivatives of these functions. If the surface is piecewise differentiable, then the maximum derivative of each of the pieces may be used. If the user does not require a guarantee about the bounding volume, then the Lipschitz constant may be estimated from samples and path lengths.

2.4 A Recursive Subdivision Mechanism

The problem of rendering deformed, intersecting surfaces is split into two subproblems: 1) the subdivision mechanism, determining how to subdivide a surface into simple elements; 2) the subdivision criteria, determining where additional sampling is necessary. Here we examine the recursive subdivision mechanism for adaptively sampling a set of surfaces.

An ideal recursive subdivision mechanism should smoothly adjust the sampling frequency over the parametric surface while adapting to variable sampling requirements. A quadtree technique is one approach for a subdivision mechanism (see Figure 2.11). The parametric (u, v) space of a surface is broken into a set of regions. Whenever more accuracy is needed in the surface quadtree, a region is divided into four smaller regions. Samples are obtained at the corners of each region.

2.4.1 Basic Approach

The subdivision technique uses the following steps:

1. The surface is sampled on a uniform parametric grid at some initial resolution.

2. Each region is evaluated using several acceptance criteria.

3. If the region is not acceptable, then it is split into four smaller regions.

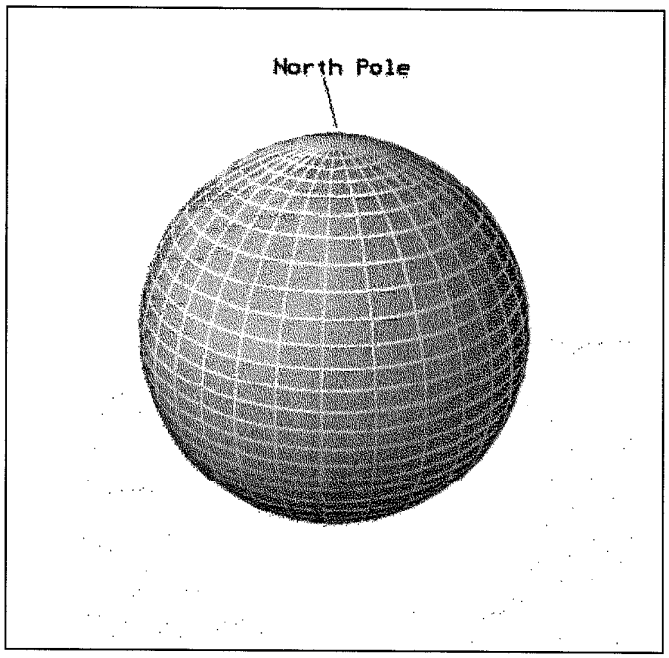

Figure 2.8: Uniformly sampled sphere.

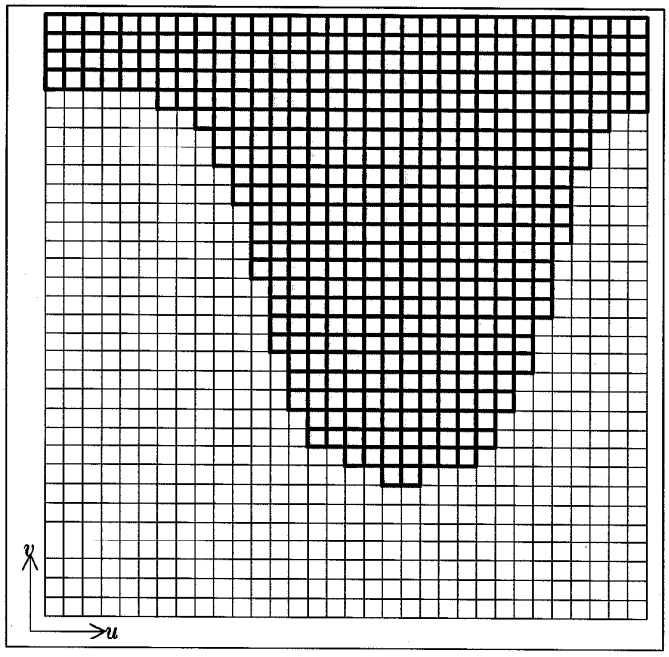

Figure 2.9: Parametric space of the uniform sphere in Figure 2.8.

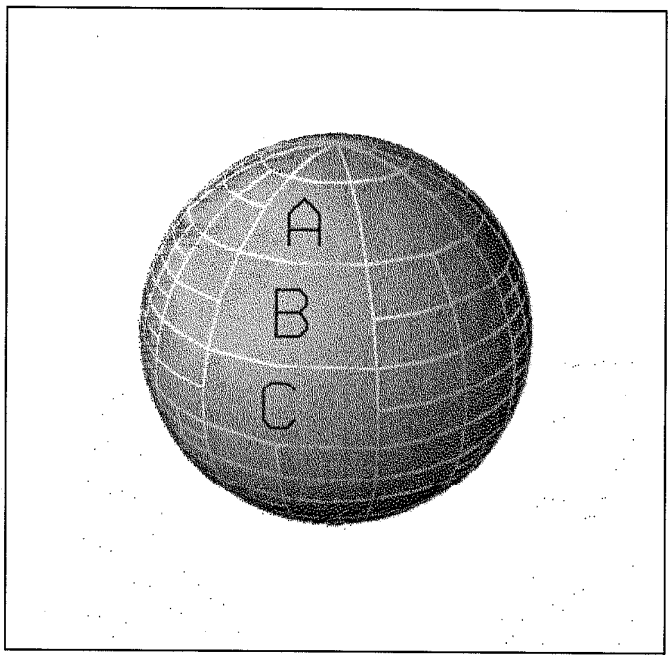

Figure 2.10: Quadtree sphere.

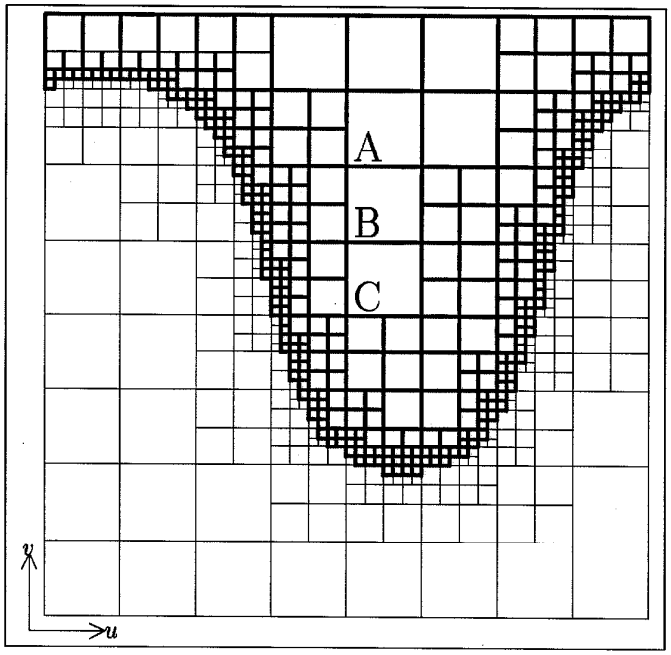

Figure 2.11: Parametric space of the quadtree sphere in Figure 2.10.

4. Steps 2 and 3 are repeated until the entire surface is adequately sampled.
5. The regions are broken into triangles.
6. The triangles are clipped at the intersection with other surfaces, forming a smooth boundary.
7. The triangles are rendered.

Figure 2.8 shows a uniformly sampled sphere, as determined by the parametric function

$$\vec{f}(u,v) = \begin{pmatrix} \cos(2\pi u)\sin(\pi v) \\ \sin(2\pi u)\sin(\pi v) \\ -\cos(\pi v) \end{pmatrix}. \tag{2.20}$$

Lines are drawn on the surface of the sphere to show how it has been broken into polygons. The surface has been sampled at the corner of each square. The north pole has more samples than necessary. Figure 2.9 shows the parametric space (u, v) of the sphere.

Figure 2.10 shows a sphere sampled using the quadtree technique. The sampling density has been increased along the boundary of the sphere, as seen from the point of view in Figure 2.10, and has been reduced in the middle and on the back side of the sphere. Figure 2.11 shows the parametric space of the sphere, after adaptive sampling. The sinusoidal line of small squares represents the silhouette boundary of the sphere, where more samples are needed to reduce visual artifacts of the sampling process. The north pole of the sphere has fewer samples than with uniform subdivision.

2.4.2 Data Structure for Quadtrees

The quadtree consists of pointers to regions arranged in a hierarchy that tessellate the parametric space of the surface. Samet describes basic data structures and access procedures for quadtrees ([Samet 82], [Samet 84]). We attach a parametric sample to the standard quadtree data structure at the corner of each quadtree region. Several regions access the same corner sample, so the samples are stored in a two-dimensional bucket array for easy access.

2.4.3 Quadrilaterals vs. Triangles

Care must be taken in creating a polygon mesh from a quadtree. Adjacent squares in a quadtree frequently vary in size. Since the corners of these adjacent squares do not always match up, cracks appear in the surface wherever small squares adjoin

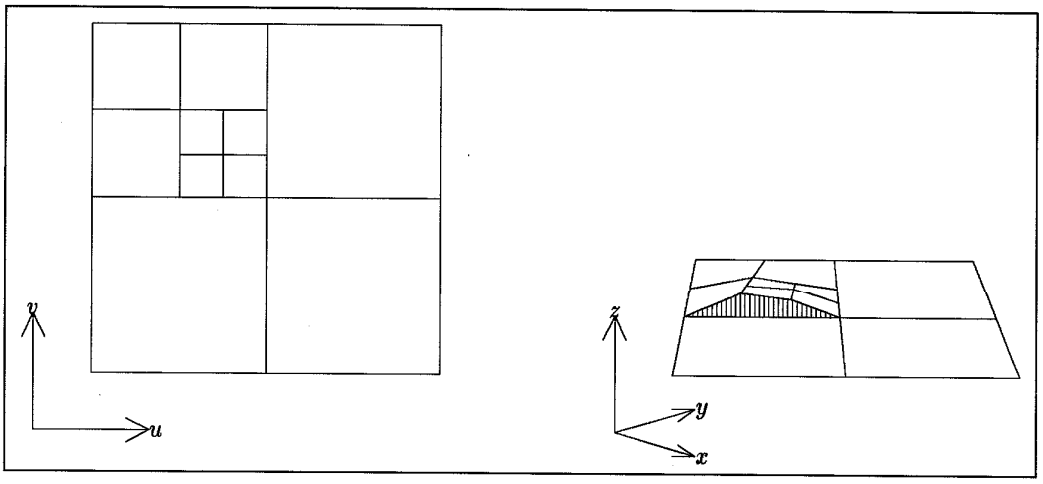

Figure 2.12: Cracks in a surface quadtree.

larger squares (see Figure 2.12). To eliminate cracks, it is necessary to construct a planar subdivision of the parametric space ([Kirkpatrick 83]). A planar subdivision is a collection of line segments that intersect only at segment endpoints. A triangular subdivision (Figure 2.13) is a planar subdivision containing only triangles. Although a quadtree is not a planar subdivision, since corners touch edges, a triangular subdivision may be constructed from the quadtree, as shown in the next section.

2.4.4 Restricted Quadtrees

The adaptive sampling process relies on the limited sampling information available to decide whether to obtain additional samples. One indication to subdivide is that the neighboring regions have subdivided for some reason. A new type of quadtree, called a restricted quadtree, propagates the subdivision information to neighbors.

The rule about restricted quadtrees is that neighboring regions must have the same width to within a factor of two (Figure 2.4). For each subdivision level in the quadtree hierarchy, the widths of the squares decrease by a factor of two. Therefore, the neighboring regions must be within one level of each other in the quadtree hierarchy. Regions that share an edge are considered neighbors. Regions that share only a corner are not considered neighbors. The rule prevents sudden changes in the sampling rate over a surface. Artifacts associated with the change in sampling rate are minimized. Restricted quadtrees concentrate samples near important features, making the algorithm more robust. The robustness of curve-finding algorithms is improved as well.

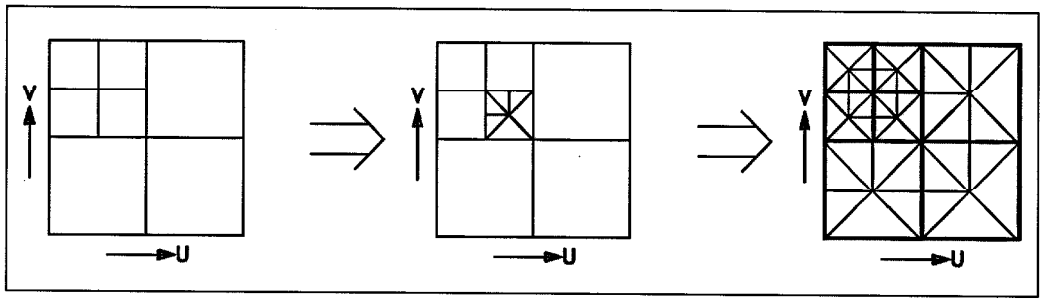

Figure 2.13: Transforming a restricted quadtree into a triangular subdivision.

Figure 2.3 and Figure 2.4 show the difference between an unrestricted quadtree and a restricted quadtree sampling near a cubic curve. The squares subdivide only if their corners span the cubic curve. The unrestricted quadtree misses a large portion of the curve, but the restricted quadtree is much more robust at exploring the complete curve.

A square in a restricted quadtree is decomposed into triangles using a simple rule. Every square is broken into eight triangles, or two triangles per edge, unless the edge borders a larger square. In that case, a single triangle is formed along the edge (see Figure 2.13). This rule does not create the minimum number of triangles per square; but it does cause most of the triangle edges to be parallel to the parametric directions u and v. For cases where the principal directions of curvature are aligned with the parametric directions, the rule produces sets of triangles that represent the surface more accurately.

2.4.5 Neighbor-Finding Algorithm

An efficient technique exists for finding neighbors in a quadtree ([Samet 82]). The algorithm finds the nearest common ancestor between a square and its neighbor, and requires an average of four node traversals of the quadtree, for quadtrees of arbitrary size. The algorithm is used to maintain the restricted quadtree, and to triangulate the quadtree. Alternatively, the neighbors may be explicitly stored with pointers at each square; this requires additional memory.

2.4.6 Parametric Space Wrap-Around

For a closed parametric surface, such as a sphere, the east boundary must match the west boundary exactly; otherwise cracks may appear at the "date line." The neighbor-finding algorithm may be extended at parametric boundaries. We define squares on the west edge of the quadtree to be neighbors of squares on the east

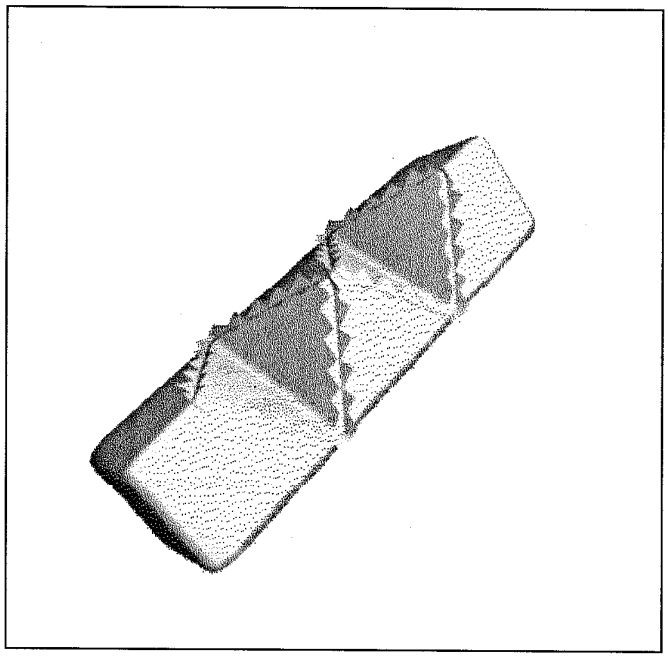

Figure 2.14: Uniform sampling without triangle clipping at surface intersections.

edge of the quadtree. For toroids, the squares on the north edge of the quadtree are neighbors of squares on the south edge of the quadtree. The seam at the parametric boundary is eliminated by forming a triangular subdivision across the boundary.

2.4.7 Triangle Clipping

After the squares of the quadtree are broken into triangles, the triangles are tested against inside-outside functions of other surfaces, and are clipped at the intersection boundary. The clipping removes the ragged appearance that otherwise occurs (Figure 2.14). Figure 2.15 shows the effects of clipping boundary triangles and quadtree sampling. The technique dramatically improves the quality of the images.

2.5 Recursive Subdivision Criteria

A set of recursive subdivision criteria is needed to determine where subdivision should occur. The philosophy of the method is to mathematically measure the

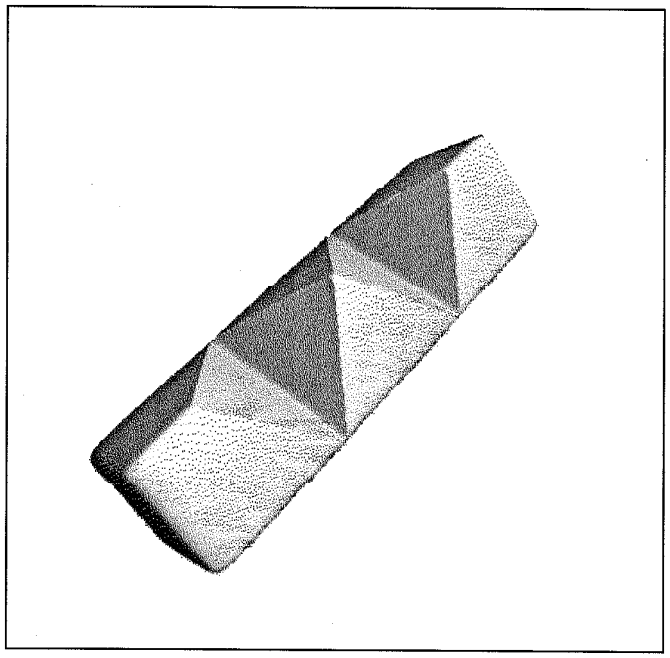

Figure 2.15: Quadtree sampling with clipping at surface intersections.

visible artifacts or errors of each part of the surface, and to subdivide until a prescribed tolerance is reached. The criteria should include a method to detect surface curvature and locate silhouette and surface intersection boundaries.

We use three coordinate systems here: (u,v) parametric space, (x,y,z) modeling space, and (X,Y) screen space. Parametric space spans the domain of the parametric surface. The surface is embedded in three-dimensional modeling space. Screen space uses a viewing transformation to project modeling space coordinates onto the image plane. Screen space is useful for determining the visual size of a feature when a model is resampled for each frame of an animation ([Barr 86]).

The recursive sampling process is started with a coarse initial grid of samples. The grid provides basic information about the surface for making decisions about further subdivision. The following criteria control the subdivision process:

2.5.1 "Curvature" Subdivision

Curvature subdivision estimates the local curvature of an object. Where the curvature is high, a region is subdivided. The subdivision process terminates when a region becomes sufficiently planar. Curvature estimation may be performed in several ways. [Lane and Carpenter 79] measure the distance from a surface to its

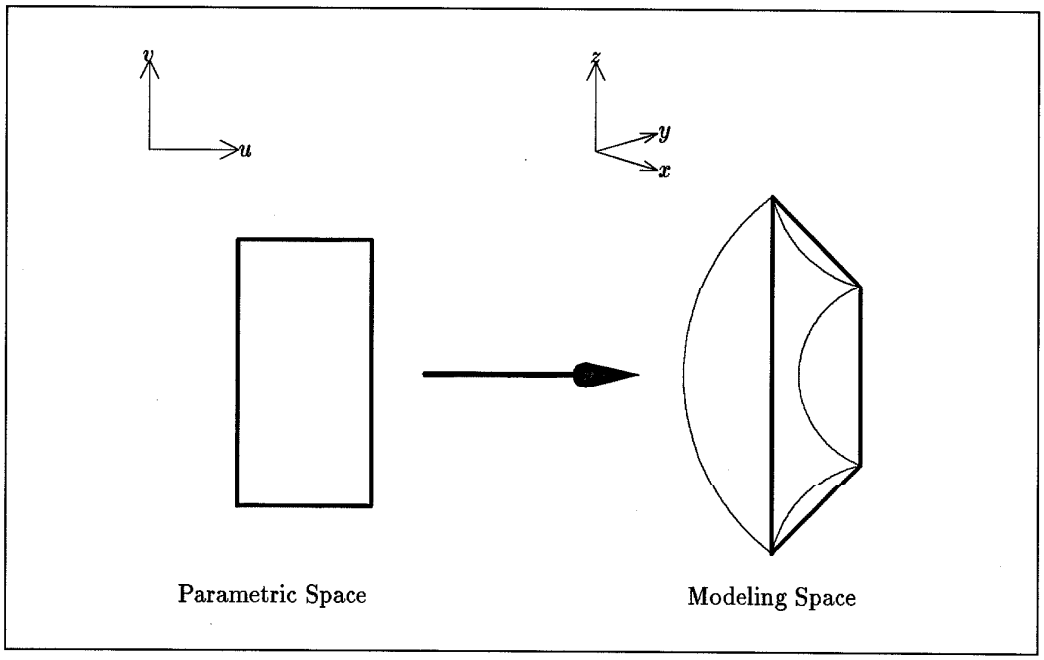

Figure 2.16: Bending a rectangle in the plane into a horseshoe shape. The quadrilateral is a very poor approximation to the bent shape.

planar approximation. Alternatively, normal vectors may be computed approximately or analytically from the equation for the surface. Normal vectors are used here, since they are computed for shading computations, anyway. A simple vector equation of these normals provides a curvature subdivision criterion. Every adjacent pair of normal vectors $(\vec{N}_1, \vec{N}_2)$ of a region must satisfy $(1-\vec{N}_1 \cdot \vec{N}_2) < \epsilon$, where ϵ is determined empirically by adjusting ϵ until the image quality is satisfactory. The normal vectors are normalized to unit length. Subdivision stops if the region is smaller than a pixel. The actual curvature k is given by $k = (d\theta/dx)$, where θ is an angle and x is a distance. For small values of θ, the normal vector estimation $(1 - \vec{N}_1 \cdot \vec{N}_2)$ computes a term proportional to θ^2.

Tangent vectors must pass the same curvature test, $(1-\vec{T}_1 \cdot \vec{T}_2) < \epsilon$. It is possible for all of the normal vectors to point in the same direction, but the tangent vectors may point in different directions. Distorting a rectangle into a U-shape is a good example (Figure 2.16). The sheet stays in the plane, but its tangent vectors are not parallel. Such highly curved regions must be sampled finely. Tangent curvature subdivision eliminates the problem (Figure 2.17), improving the robustness of the curvature subdivision criterion.

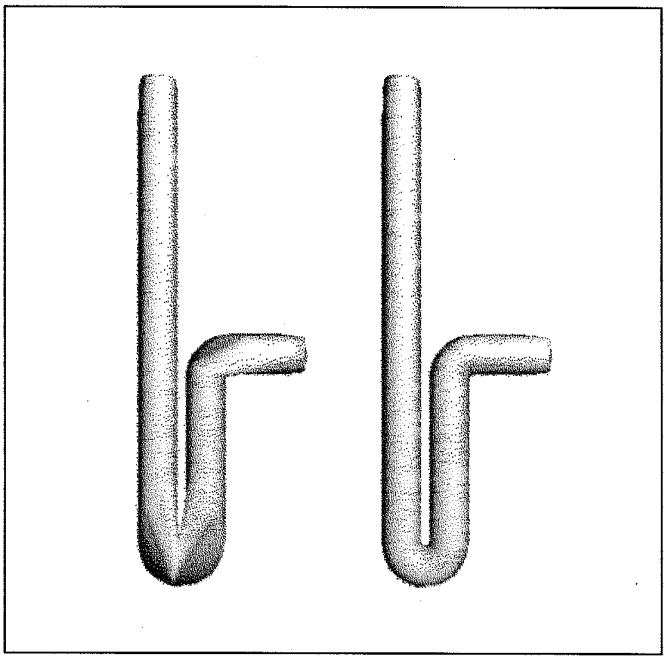

Figure 2.17: On the left is a drain without tangent curvature subdivision. The image on the right shows a drain with tangent curvature subdivision. Note the additional sampling near sharp corners.

2.5.2 Intersection Subdivision

A sharp boundary is created where surfaces intersect. The boundaries should be finely sampled to avoid aliasing artifacts. One approach is to subdivide boundary regions until they are smaller than a pixel. The technique is robust, but expensive, in computing the boundary of the surface. An alternative technique is explored here that measures the straightness of the boundary. Subdivision occurs until the boundary is straight in modeling (x, y, z) space. In regions where the boundary forms a sharp corner, subdivision stops if the region is smaller than a pixel. Where the boundary is straight, larger triangles can be used. Triangle clipping at the boundary produces an accurate boundary if the boundary is straight.

The test for straightness of a boundary curve across a region uses approximate tangent vectors of the boundary curve. Figure 2.18 shows a square with four corner vertices connected to a center vertex. Note that any straight line crossing the square must intersect the lines of the square in at least three places: $\vec{P}_1$, $\vec{P}_2$, $\vec{P}_3$. The intersection points are found by interpolating between corner samples and a center sample on the square until a boundary point is obtained. A variation

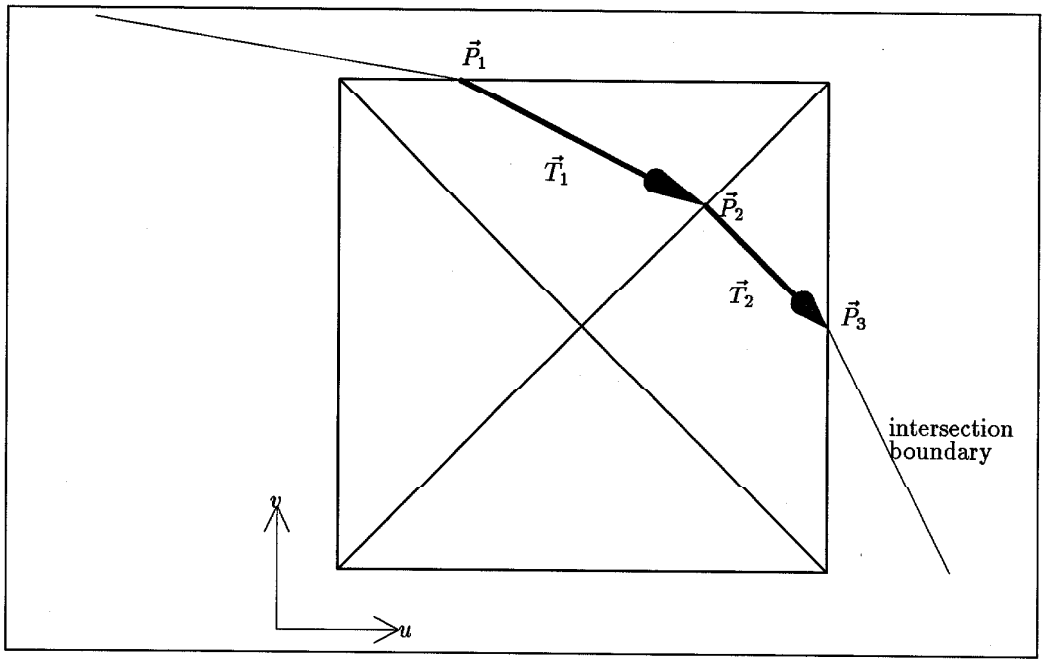

Figure 2.18: Geometry for intersection boundary subdivision.

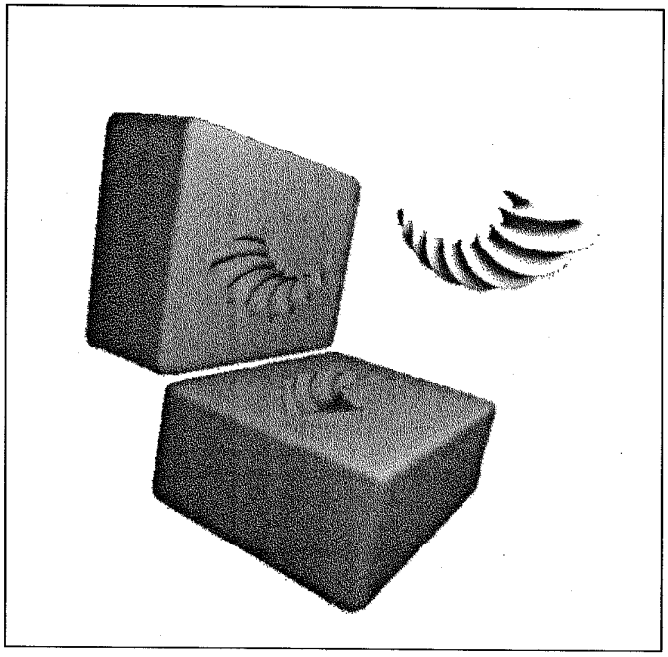

Figure 2.19: Subtraction of a deformed surface.

of the *regula falsi* iteration method is effective ([Rao 84]). We approximate the boundary by the line segments $\vec{T_1} = \vec{P_2} - \vec{P_1}$ and $\vec{T_2} = \vec{P_3} - \vec{P_2}$. The vectors $\vec{T_1}$ and $\vec{T_2}$ are approximate tangents to the boundary curve. The two tangent vectors are normalized and tested with the condition $(1 - \vec{T_1} \cdot \vec{T_2}) < \epsilon$. The region is subdivided until the condition is met, unless the region is smaller than a pixel in screen space. Triangle clipping smooths the edges created.

Several methods exist for telling whether points are inside or outside an intersecting object. A hierarchical set of bounding volumes can determine the proximity of a sample point to the intersecting surface. Alternatively, an inside-outside, or implicit, formulation for the surface can classify the sample point ([Barr 83], [Sederberg and Parry 86]). Figure 2.19 shows an example of a deformed surface subtracted from a mold.

2.5.3 Silhouette Boundaries

Given a camera viewpoint, silhouette subdivision concentrates samples along the silhouette boundary to minimize local artifacts. The eye is quick to pick up slight irregularities at the sharp border of an object, which has high spatial frequencies. Polygonal artifacts are easier to see near the silhouette boundary of a surface than on the interior. The silhouette criterion is evaluated in a similar manner to the intersection subdivision criterion. The sphere in Figure 2.10 and Figure 2.11 demonstrates the silhouette subdivision process.

The dot product between the surface normal N and the view vector V determines whether a sample is front-facing ($N \cdot V < 0$), back-facing ($N \cdot V > 0$), or on the silhouette boundary ($N \cdot V = 0$). Subdivision stops when the curvature of the silhouette boundary in screen space is less than a threshold value or when the region is smaller than a pixel. The second termination condition prevents sharp corners from causing infinite recursion.

2.5.4 Proximity Subdivision

Proximity subdivision searches for intersection points between surfaces. It is a precursor to the intersection criterion, which finds the entire intersection boundary, given an intersection point. A surface is subdivided until either an intersection is found or a local minimum is found in the inside-outside functions of the other surfaces. If an implicit inside-outside function is not available, bounding volumes or surface regions may be used to determine the need for additional subdivision.

2.5.5 Efficient Combination of Subdivision Criteria

Each of the subdivision criteria described above is computed for each region. Sometimes it is possible to determine that a region requires further subdivision without computing all the criteria. In this case, it makes sense to compute the inexpensive criteria first so as possibly to avoid unnecessary computation. When the region passes all subdivision criteria once (or is culled from further consideration), a flag is set to indicate that the region should not be reexamined during the next pass through the quadtree. Some of the criteria use view-dependent tests, since a viewing transformation may be available at sampling time. In addition to the criteria mentioned here, the region may be forced to subdivide due to the constraint of restricted quadtrees that neighboring regions remain the same width within a factor of two. The following tests are ordered roughly according to increasing computational cost:

1. If the square is bigger than the initial sampling grid, then subdivide.
2. Else, if the square is facing away from the viewer, then stop subdividing.
3. Else, if the square is smaller than a pixel, then stop subdividing.
4. Else, if the proximity test reveals a potential intersection, then subdivide.
5. Else, if the square is culled by a surface intersection, then stop subdividing.
6. Else, if the square fails the flatness test, then subdivide.
7. Else, if the square fails the intersection boundary curve straightness test, then subdivide.
8. Else, if the square fails the silhouette boundary curve straightness test, then subdivide.
9. Else, stop subdividing the square.

2.6 Imaging Results

The following illustrations show the variety of deformed, intersecting parametric surfaces that may be rendered using the adaptive sampling techniques described.

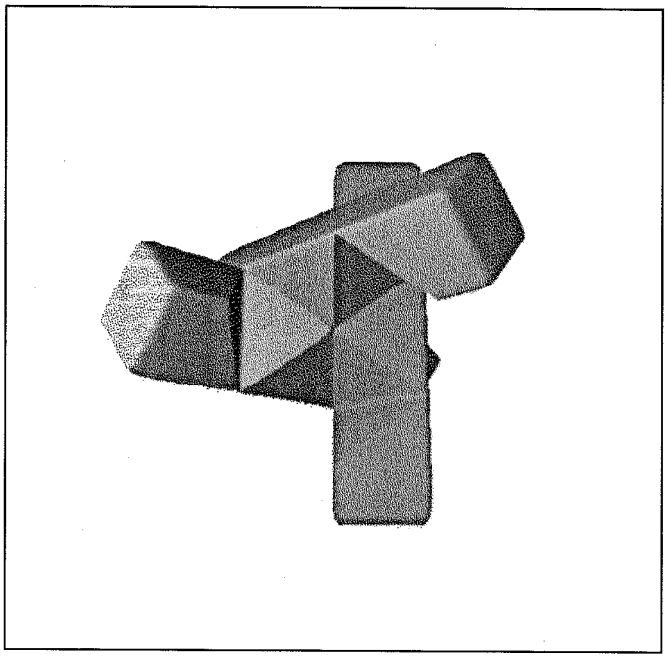

Figure 2.20: Triple cluster.

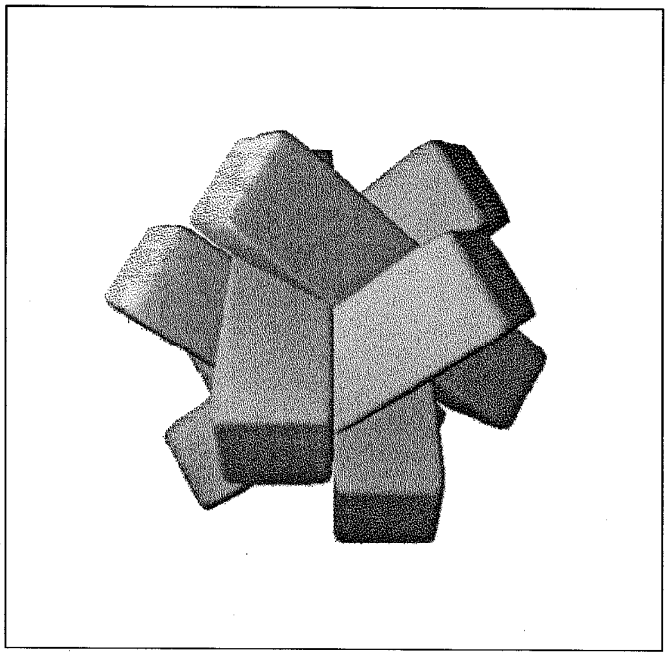

Figure 2.21: Hex puzzle.

Figure 2.22: Bicycle chainwheel.

2.6.1 Puzzle

The puzzle is modeled with six identical pieces that fill the volume of the interior of the puzzle. An individual piece is formed by taking a superquadric block, and subtracting two similar blocks to cut wedges in the piece (Figure 2.15). Three pieces may be assembled to form the cluster shown in Figure 2.20. The completed puzzle is formed with a left-handed triple cluster and a right-handed triple cluster (Figure 2.21).

2.6.2 Bicycle Chainwheel

The chainwheel uses the Boolean subtraction operation extensively (Figure 2.22). Fifty-two cylinders are subtracted from a disk to form the teeth for the gear. The proximity subdivision criterion helps to locate the position of the teeth. Deformed cylinders cut holes in the gear to make it lighter. Superquadric cranks and pedals are added after the cutting process.

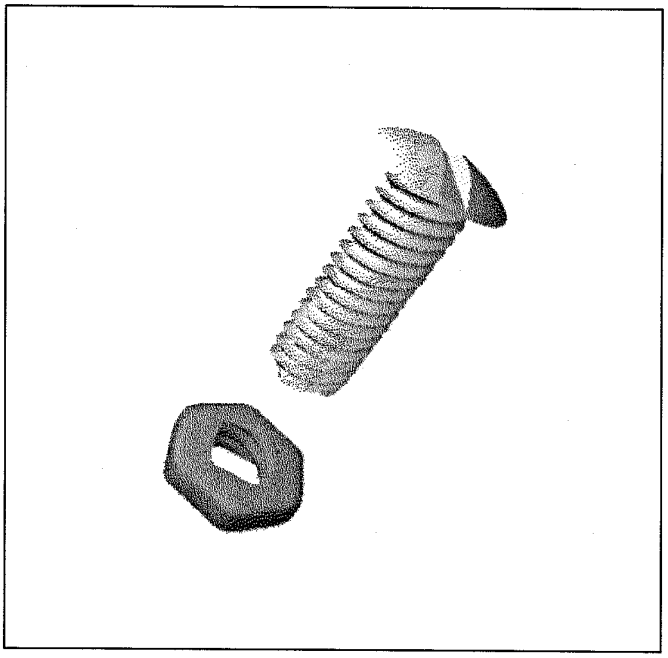

Figure 2.23: Nut and bolt.

2.6.3 Nut and Bolt

A type of screw thread may be formed by taking a superquadric with a square profile and twisting it for several revolutions (Figure 2.23). The thread is subtracted from the nut and merged with the bolt head to form the nut-and-bolt combination.

2.7 Summary

Surface quadtrees are an effective way to triangulate deformed, intersecting parametric surfaces. The adaptive sampling problem may be decomposed into two subproblems: the mechanism for subdivision, and the subdivision criteria. Images of these parametric surfaces have been created using a robust subdivision mechanism and a small set of subdivision criteria.

Restricted quadtrees are more robust than unrestricted quadtrees in the triangulation of parametric surfaces. Curvature, intersection, proximity and silhouette subdivision techniques provide a robust set of criteria for recursively sampling parametric surfaces. These techniques are found to be more efficient than uniform subdivision at producing triangulations of deformed, intersecting surfaces.

Chapter 3

Subdivision Mechanisms for Adaptive Parametric Sampling

This chapter describes a series of subdivision mechanisms that adaptively sample parametric surfaces. The purpose of an adaptive subdivision mechanism is to smoothly adjust the sampling frequency across a surface, as required by various sampling criteria. It is important that the subdivision mechanism not produce cracks or other artifacts in the surface to be represented. It is also desirable to have some control over the aspect ratios of surface elements so that sampling may be increased along one parametric dimension independent of the other dimension. Finally, the results should extend to functions of an arbitrary number of parametric variables.

A variety of subdivision mechanisms are possible that may satisfy the objectives. Here we describe and compare a progression of subdivision mechanisms in an effort to refine subdivision techniques.

3.1 Uniform Sampling

The simplest way to sample a surface is with uniform sampling (Figure 2.9). The samples are not focused on regions of interest, but are distributed uniformly across the surface. The complexity of this approach is $O(n^2)$ samples for linear resolution n.

3.2 Unrestricted Quadtrees

The quadtree data structure has been extensively explored by [Samet 84]. The quadtree uses a basic recursive subdivision process whereby a square is subdivided

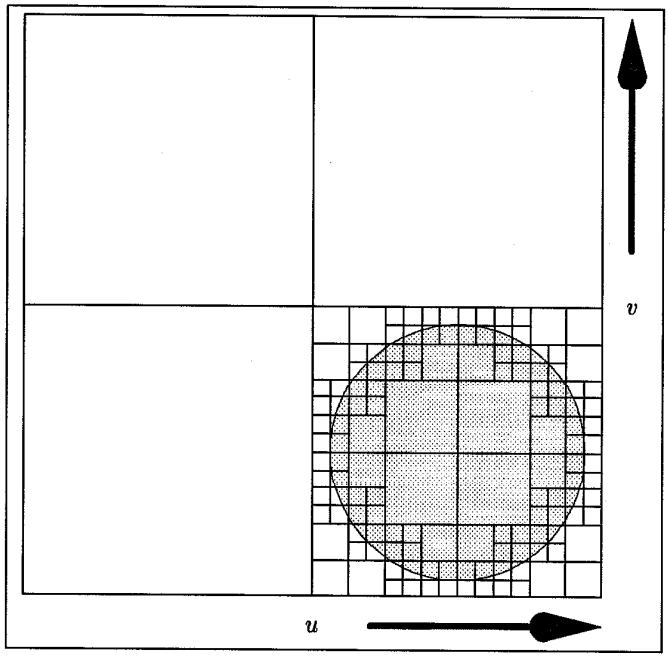

Figure 3.1: An unrestricted quadtree of a circular region showing squares with a large number of neighbors. It is difficult to construct a satisfactory triangular subdivision from this structure.

into four smaller squares. This basic approach is simple, but suffers from several deficiencies. The quadtree is subject to abrupt changes in sampling frequency across a surface, resulting in frequent omissions of features that are important to sample. In Chapter 2, we showed that if corner sampling is used with quadtrees, it is common to lose linear features on the scale of the square being sampled (Figure 2.3).

Another problem with quadtree squares is that they are hard to convert into suitable triangles. A quadtree square may have an arbitrary number of smaller neighbors Figure 3.1. It is difficult to construct a triangular subdivision out of this configuration that avoids cracks in the surface, avoids lots of additional sampling across the square, and preserves the aspect ratio of the resulting triangles. Other subdivision mechanisms have been shown to have advantages over the basic quadtree method [Von Herzen 85].

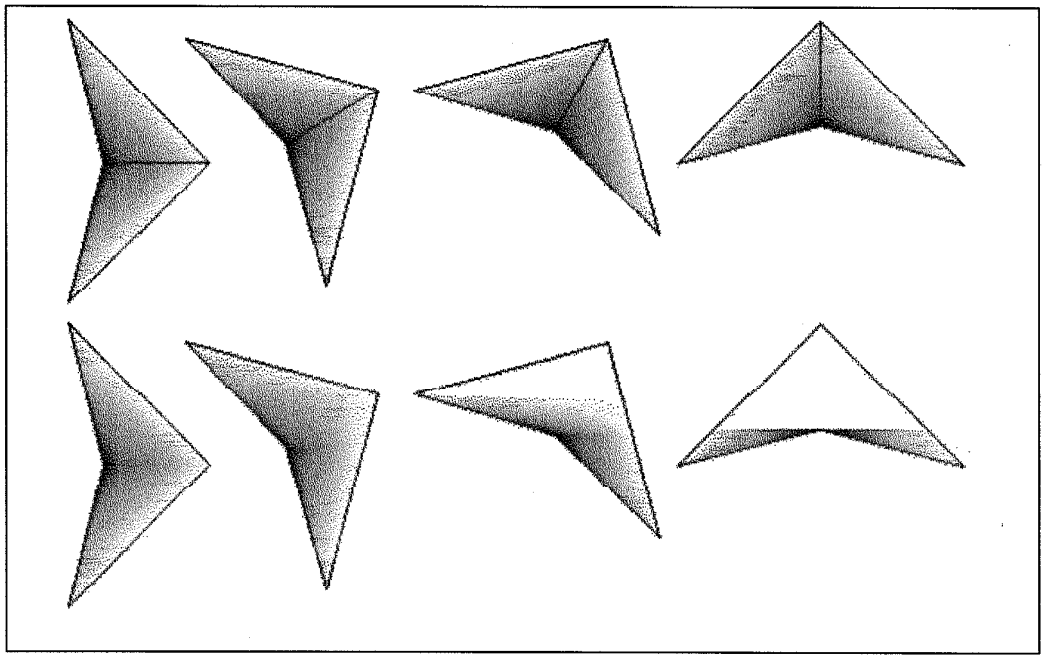

Figure 3.2: Shading that uses scanline interpolation, such as Gouraud shading, is rotation-invariant only for triangles (top row). Gouraud shading with other polygons produces serious artifacts (bottom row).

3.3 Restricted Quadtrees

Chapter 2 developed the notion of a restricted quadtree as an attractive mechanism for the adaptive sampling of parametric surfaces. The procedure involves the recursive subdivision of parametric space using quadtrees, subject to the restriction that neighboring squares in the quadtree may differ in width by at most a factor of two (Figure 2.4). If the neighbor size is restricted, then additional subdivision is performed near regions with a high concentration of samples, resulting in a smooth transition from low spatial sampling frequencies to high ones. Restricted quadtrees are much more robust than unrestricted quadtrees at adequately sampling curves and surfaces. Figure 2.11 shows a restricted quadtree approximating the silhouette of a sphere in parametric space.

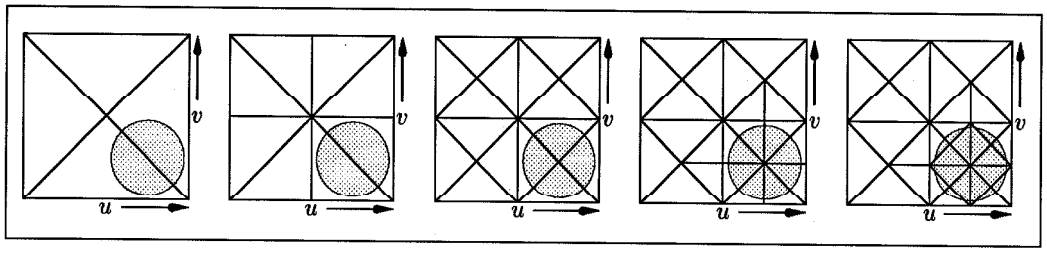

Figure 3.3: A surface network of right triangles, generated directly from a right triangular subdivision mechanism.

3.4 Advantages of Triangular Surface Elements

Restricted quadtrees are converted into triangles because triangular surface elements offer several advantages over other surface elements. Polygon renderers usually interpolate shading across the interior of a polygon. Many shading algorithms ([Gouraud 71]) are rotation-invariant only for triangles (Figure 3.2). In fact, if interpolation is done horizontally, as for Gouraud shading, the shading pattern becomes discontinuous for polygons that are concave in the vertical dimension (See the last quadrilateral in Figure 3.2). We would like to guarantee that all polygons sent to the rendering system are convex. If we use parametric quadrilaterals, we have no assurance that the image of the quadrilateral in modeling space will remain convex. Triangles are never concave, so the problem never arises if we use parametric triangles.

Triangles are useful in forming a *planar subdivision* of a restricted quadtree. One way to avoid cracks in a surface is to form a planar subdivision of the parametric space. A planar subdivision is a collection of line segments that intersect only at segment endpoints. A triangular subdivision of the plane may be constructed from the restricted surface quadtree (Figure 2.13).

Triangles are advantageous in the computation of surface normals as well. Three non-degenerate vertices are necessary and sufficient to determine the normal for the triangle. With four or more vertices, we may not have a unique normal vector if all four vertices are not coplanar. This contradiction is not possible for triangles; the data structure is correct by construction.

For these reasons, the work here focuses primarily on triangular surface elements rather than on more complicated polygonal primitives.

Figure 3.4: An image of a silhouetted figure.

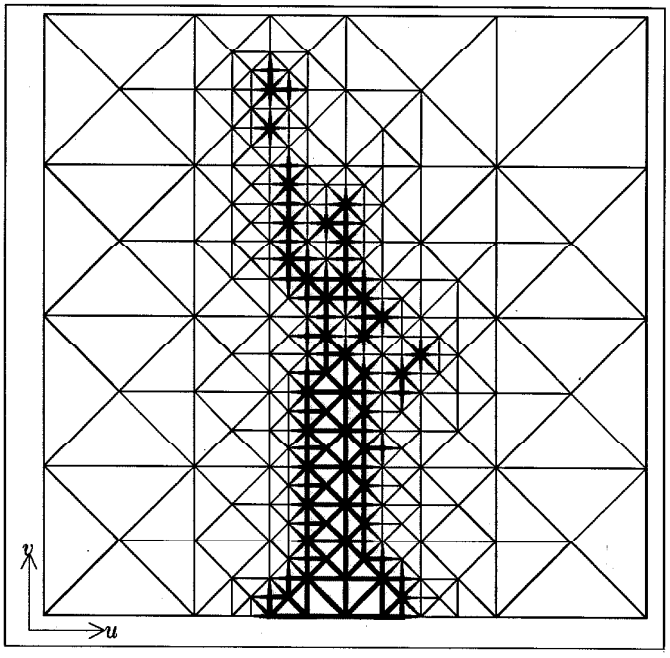

Figure 3.5: A right triangular surface network of an image of a silhouetted figure.

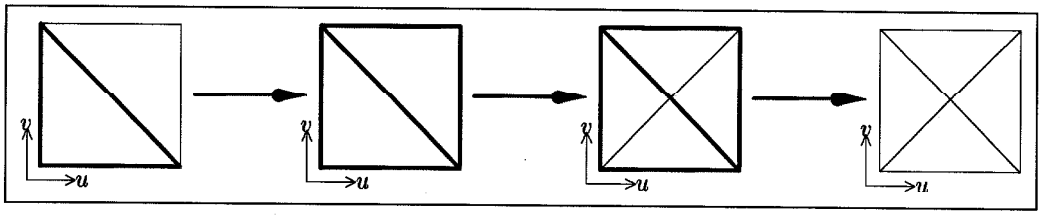

Figure 3.6: The bold triangle must be split. We split along the hypotenuse, and must split the neighbor as well, to maintain a right triangular subdivision.

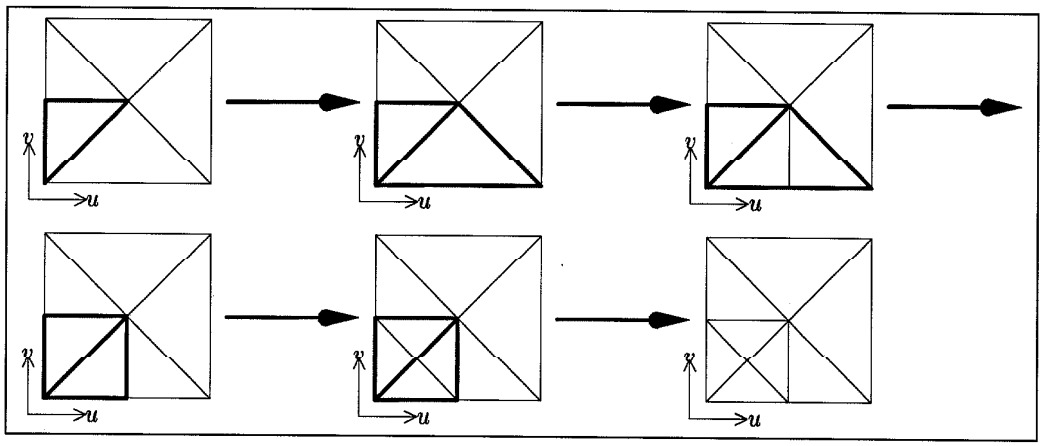

Figure 3.7: The bold triangle must be split. We move to larger neighbors until we can find a pair as in Figure 3.6.

3.5 Bintrees of Right Triangles

The restricted quadtree subdivision method recursively subdivides a square parametric region until it is sufficiently small, and then converts the region into a set of triangles. It may be more efficient to start with triangles in the first place, and recursively subdivide them until they reach adequate size. This line of reasoning leads to the notion of bintrees of triangles, as opposed to quadtrees of squares. Figure 3.3 show a simple subdivision mechanism based on right triangles. The hypotenuse is always split, forming two legs of smaller triangles. This recursion mechanism has advantages similar to restricted quadtrees, but is more efficient than restricted quadtrees in that fewer samples are required to obtain the same spatial resolution (See Appendix B.4). Figure 3.5 shows a silhouette figure, sampled using bintrees of right triangles.

In order to avoid cracks in the surface, the triangular bintree must span the parametric region, and must form a triangular subdivision. If we split a single

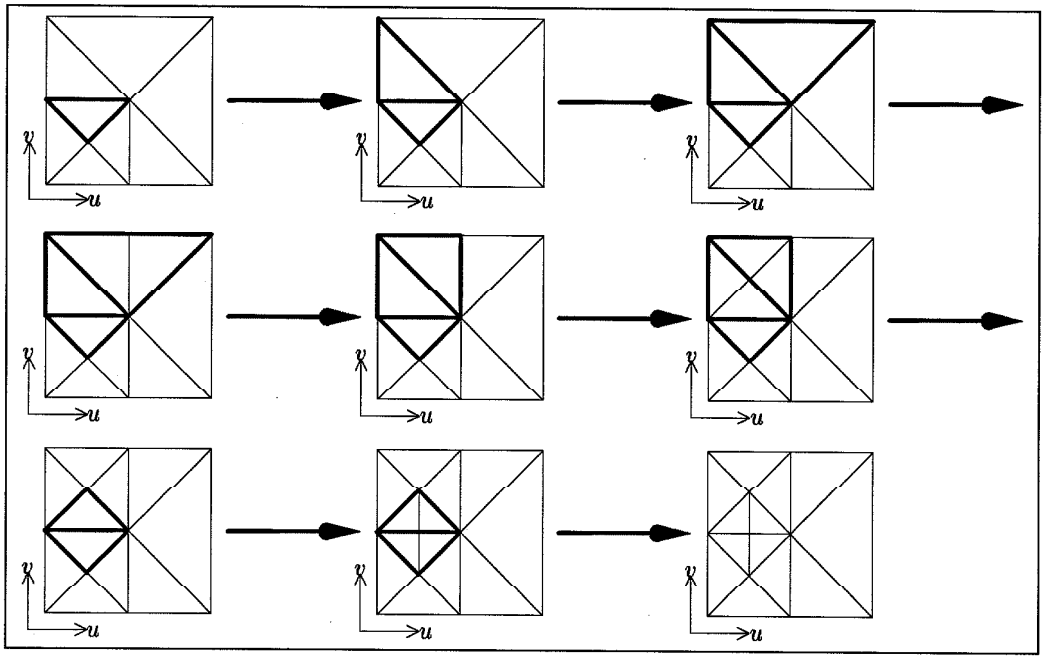

Figure 3.8: A three-step recursion to subdivide the bold triangle. The recursion terminates because we always go to larger triangles.

triangle without splitting its neighbor, we would destroy the requirement that triangle vertices always touch other vertices, never other triangle edges. If we always split pairs of triangles with a common hypotenuse, we preserve the triangular subdivision (Figure 3.6). If the hypotenuse of the split triangle matches the leg of the neighboring triangle, then we must subdivide the neighbor first along the hypotenuse, and then along the leg (Figure 3.7). In some cases we must recurse to larger and larger triangles to find a pair with a common hypotenuse. Since we always proceed from smaller triangles to larger ones in this recursion, either the recursion terminates or we reach the largest triangle in the surface network, in which case the recursion stops. Edge triangles can split by themselves without destroying the triangular subdivision, as long as they aren't connected to other surface networks. Figure 3.8 shows a multiple step recursion to subdivide a small triangle.

The implementation of the triangular bintree subdivision process is quite simple. Only one primitive operation alters the triangular subdivision. The operation transforms a pair of triangles with a common hypotenuse into four triangles with common legs, as in Figure 3.6.

In effect, we are allowed to split a triangle only along its longest edge. This

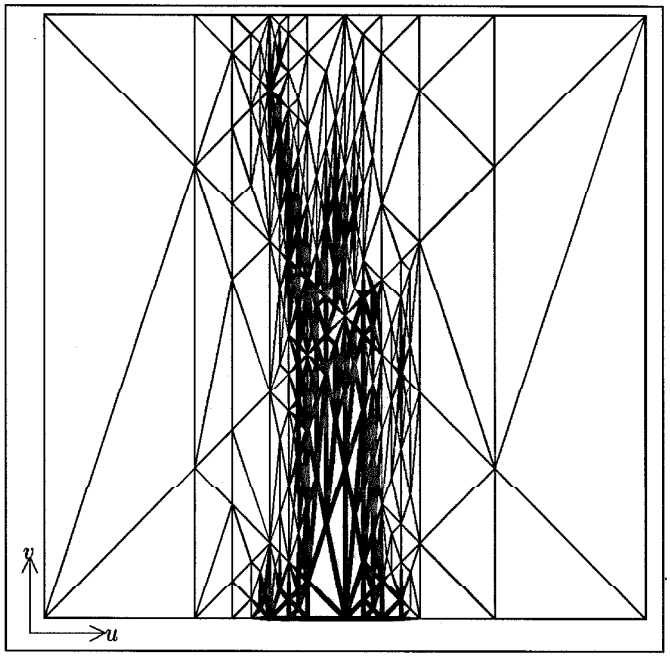

Figure 3.9: A scaled triangular surface network of an image of a silhouetted figure.

preserves the aspect ratio of the triangles as they are split. As long as there is a total ordering of all the edges in the subdivision, the algorithm will terminate, since the recursion moves exclusively from shorter edges to longer ones. Since there is a maximal edge, termination is ensured. The length of each edge serves as the total ordering function for the bintrees of right triangles, and thus ensures termination of the recursion.

3.6 Planar Subdivisions with General Triangles

The previous section dealt with right triangles. It is possible to generalize the subdivision mechanism to deal with other types of triangles. We used edge length to choose which edge to split with binary right triangles. It is possible to use other total orderings to create subdivision schemes using non-right triangles.

3.6.1 Triangular Subdivision with a Scaled Length Metric

One simple extension of the right triangular subdivision algorithm is to use a definition of edge length stretched in one direction, such as

$$L^2 - su^2 + v^2, \tag{3.1}$$

where L is length, u and v are parametric dimensions, and s is a scaling for the u parametric direction. When $s > 1$, Eqn. 3.1 forces more subdivision in the u direction than the v direction, which is useful when sampling long, skinny surfaces, or in any situation where more sampling is needed along one parametric direction than along the other. The statue silhouette of Figure 3.4 benefits from finer sampling in the u direction in Figure 3.9, because of the vertical alignment of its edges. Since the boundary is longer in one parametric direction than in the other, this subdivision mechanism results in an improvement in efficiency over bintrees of right triangles.

3.6.2 Adjusting the Aspect Ratio of Triangular Subdivisions

Suppose that we wanted no edge of a triangular surface network to be longer than a distance D in modeling space. If some regions of the surface are long and skinny in terms of the parametric aspect ratio, $(\Delta v/\Delta u)$, then we would like to control the aspect ratio of the triangles on the surface so as to make effective use of the samples being generated. This can be done by changing the edge-ordering function to match the desired property of the subdivision. For example, if we use modeling space length instead of parametric length as the edge-ordering function, then all edges of length $L > D$ are split before all edges of length $L \leq D$. This technique uses fewer triangles than in the case of scaled triangular subdivision.

The binary triangular method will work for any total ordering function that operates on the edges of a surface network. The best ordering function to use depends on the criteria for subdivision.

3.6.3 Reorienting Triangular Subdivisions

You might think that an easy subdivision mechanism would be simply to split an edge whenever it crosses a boundary of a region. This should result in short edges near boundaries of regions, and in long edges within homogeneous regions.

Unfortunately, the technique does not work. Figure 3.10 demonstrates that the technique fails to traverse the boundary of the silhouetted figure of Figure 3.4

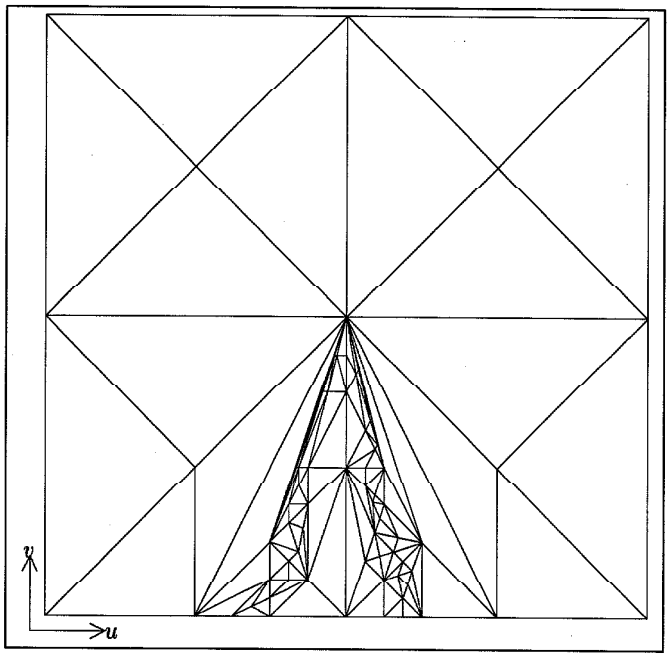

Figure 3.10: The failure of a technique that samples by subdividing edges only when they cross the boundary of a silhouette.

successfully. The triangles are unable to force sufficient subdivision of neighbors to traverse the complete silhouette.

The technique in Figure 3.11 is somewhat more successful at sampling the boundary of the figure. It is not clear that this approach is really optimal in terms of speed, robustness, or number of triangles, but it is representative of the wide variety of subdivision mechanisms that are possible.

3.7 General Tetrahedral Subdivision for Parametric Functions of Three Variables

The binary subdivision technique described in Section 3.6 can be generalized to higher dimensions quite successfully. The same binary subdivision technique will work with simplexes of any dimension. For three parametric variables (u, v, t), we must span the parametric solid with a tetrahedral subdivision.

We will still use the total ordering function for edges to determine which edge of the tetrahedron to split. Figure 3.12 shows the geometry for tetrahedral subdivision. Only one edge of the tetrahedron is split, the one that is maximal with

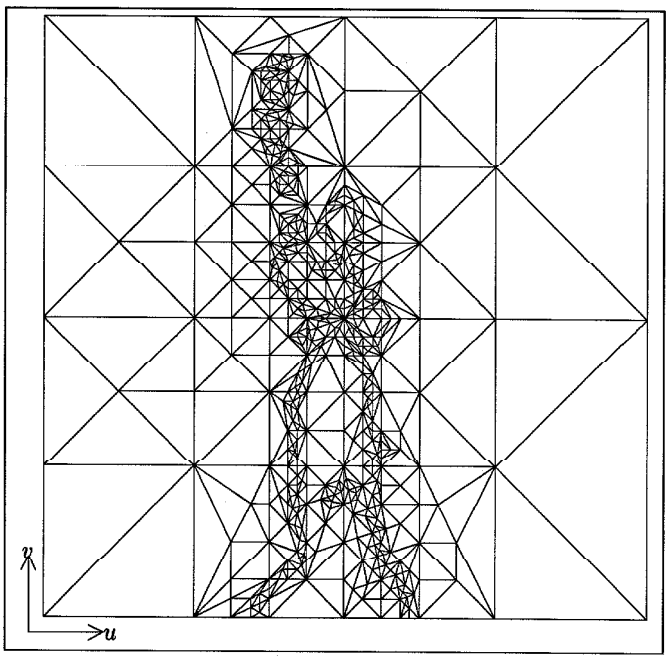

Figure 3.11: Here is an alternative subdivision mechanism using general triangles. We split all triangles that cross the boundary of the silhouette, bisecting the longest edges of the triangles. If the edge straddles a boundary, its length is multiplied by 1.46. We choose a value slightly over $\sqrt{2}$ so that right isosceles triangles sometimes split along a leg instead of a hypotenuse.

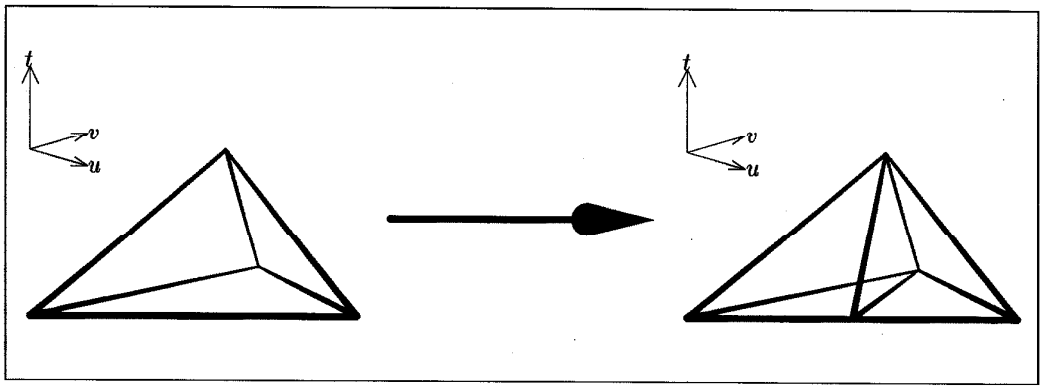

Figure 3.12: Geometry for tetrahedral subdivision. The tetrahedron is split along the longest edge, forming two smaller tetrahedra.

respect to the edge-ordering function.

All other tetrahedra that touch the maximal edge must be split as well. If they must first split in some other direction, then the splitting function is recursively called, in a manner analogous to the two-dimensional triangular case. The tetrahedral algorithm terminates for the same reasons that the triangular algorithm terminates. As long as there is a total ordering of all the edges in the subdivision, termination is ensured, since the recursion moves exclusively from shorter edges to longer ones.

This method is useful for higher-dimensional parametric problems, such as parametric surfaces moving as a function of time. The same results on avoiding cracks in modeling space apply to higher-dimensional subdivisions. The methods established here will work with simplexes of any dimension, given a total ordering of edges of the subdivision.

3.8 Non-planar Surface Elements

An interesting potential future extension of this work would be to consider using non-planar surface elements instead of triangles to approximate parametric surfaces. A difficulty with this approach is that a polygon is the most complicated primitive directly rendered by many rendering systems. One possibility is to consider a method that generates simple curved patches, and then to generate triangles from the curved patches.

Steiner patches, which are rational quadratic surfaces, have recently been used for surface rendering ([Sederberg and Anderson 86]). These are non-planar triangular patches, with no inflection points on their interior (Figure 3.13). Given three arbitrary corner points, a set of four Steiner patches may be sufficient to provide C_1 continuity across the surface. Since Steiner patches are triangular, it is attractive to use bintrees of right triangles to generate the surface samples. Since the bintree algorithm generates a triangular subdivision (no triangle edges intersect), we can guarantee both C_0 and C_1 continuity across surface patches.

Another approach is the use of biquadratic surface elements to model surface curvature as well as orientation ([Barr 86, p. 294]). The patches do not exactly match at their edges (neither C_1 nor C_0 continuity), but a new patch is generated whenever the displacement of the adjacent edges of two patches exceeds a threshold value. In this way it is possible to keep the approximation errors below a level perceptible to the human eye.

Another possible approach in using non-planar surface elements comes from research in image analysis in the segmentation of an image into smooth subelements ([Besl and Jain 88]). The approach is to determine the local Gaussian and mean

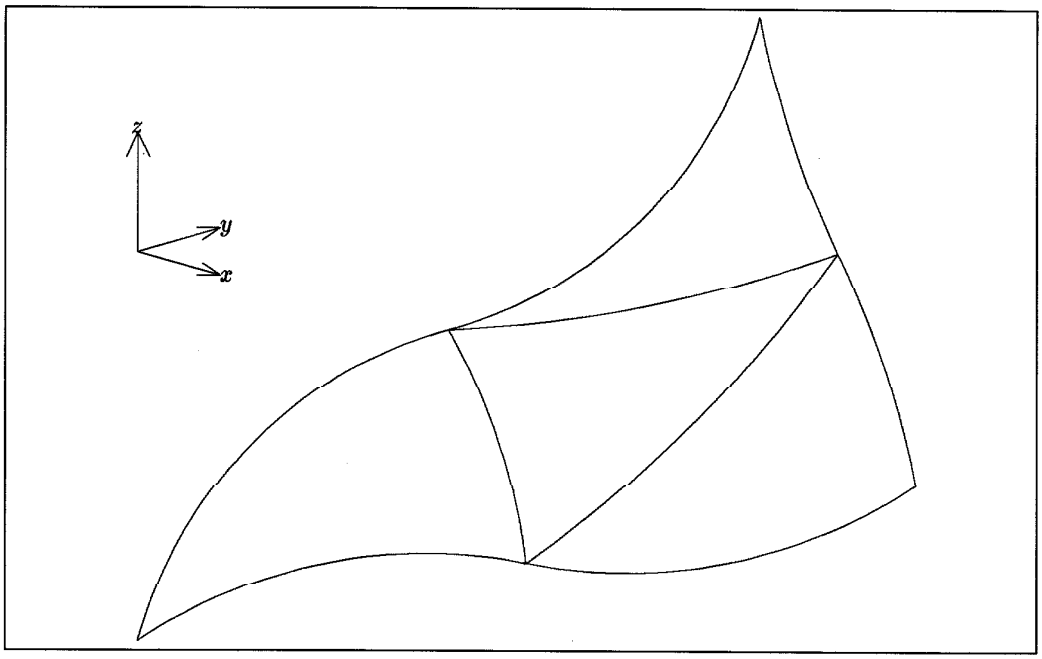

Figure 3.13: A set of four Steiner patches that forms a triangular non-planar surface element.

curvature of the intensity function of the image at each pixel, and then to grow surface patches that are biquadratic, bicubic, or biquartic in complexity, based on the geometry of the smooth features in an image. The initial assumption is made that the surfaces are as simple as possible, and when that assumption is shown to be false, the order of the approximating surface is increased in an attempt to obtain a better approximation to the surface. The order of the approximating surface is increased until the mean errors drop below a threshold value.

3.9 Summary

In this chapter we illustrate a series of subdivision mechanisms that can be used to adaptively sample a parametric surface. Triangular surface elements are shown to have important benefits over quadrilateral elements. A new method for subdivision based on bintrees of right triangles has superior efficiency to standard quadtree techniques. Variations on the basic approach using bintrees of right triangles are possible. These lead to more general triangular subdivisions, but many of these are unsatisfactory for delineating the boundaries of surface regions. Future extensions

to non-planar surface elements are discussed.

Chapter 4

Collision Determination for Parametric Surfaces

In this chapter we develop a method for computing ϵ-*collisions* between parametric surfaces for which we have bounds on the parametric derivatives of the surfaces. A collision may be defined as the loss of separation between objects. The notion of an ϵ-collision is developed, whereby a volume of width ϵ contains points from both parametric surfaces. An ϵ-collision occurs whenever a volume of width smaller than a given distance ϵ is found to contain points from both surfaces. The ϵ-collision algorithm will find all ϵ-collisions between the two surfaces. We are able to guarantee that we will find the first ϵ-collision, if one exists, to within the temporal tolerance of the computation. In cases where the parametric surfaces are far apart, the ϵ-collision algorithm terminates after a single sample has been taken from each surface. It becomes computationally trivial to reject potential collisions between distant objects. The closer the two surfaces are to each other, the longer it takes to verify that the two objects do not collide. The technique applies potentially to problems in robotics and aviation.

4.1 Introduction

4.1.1 Problem Statement

Many problems in spatial computation require the ability to determine collisions between objects. Questions of how to quickly and reliably determine collisions between complex surfaces apply to robotics, air traffic control, physically-based simulation, and modeling. This chapter addresses the problem of determining the time and position of first collision between two parametric surfaces that are moving as a function of time (Figure 4.1). The parametric surfaces may be considered to

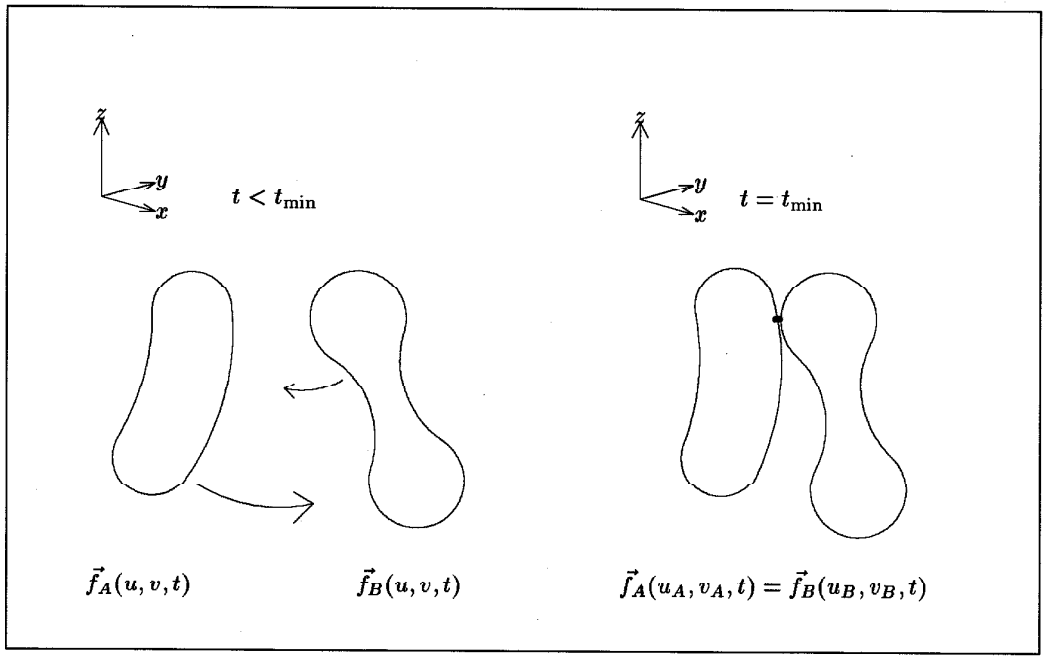

Figure 4.1: The collision-determination problem for arbitrary surfaces: to find the time and location of the first collision between two moving parametric surfaces.

be vector functions of three parametric variables: $\vec{f}_A(u,v,t)$ and $\vec{f}_B(u,v,t)$, where u and v are parametric variables that span the surfaces, and t is time. For suitable types of surfaces, we want to find the earliest time $t_{\min}$, such that

$$\vec{f}_A(u_A,v_A,t_{\min}) = \vec{f}_B(u_B,v_B,t_{\min}). \tag{4.1}$$

We also want to find u_A, u_B, v_A, and v_B at some point of first collision on each surface. We assume that the surfaces are continuous, and that they are embedded in three spatial dimensions and one temporal dimension.

4.1.2 Problems with Arbitrary Surfaces

The collision problem for parametric surfaces can be made arbitrarily difficult for suitably extreme parametric surfaces, such as the spike function of Figure 4.2. For suitably sharp spikes, finite sets of samples will probably miss the spikes completely. If we do not know where the spikes are, then we cannot know when the surface will collide with other surfaces. Finding a narrow spike becomes arbitrarily difficult as the parametric width of the spike approaches zero. There must be some additional

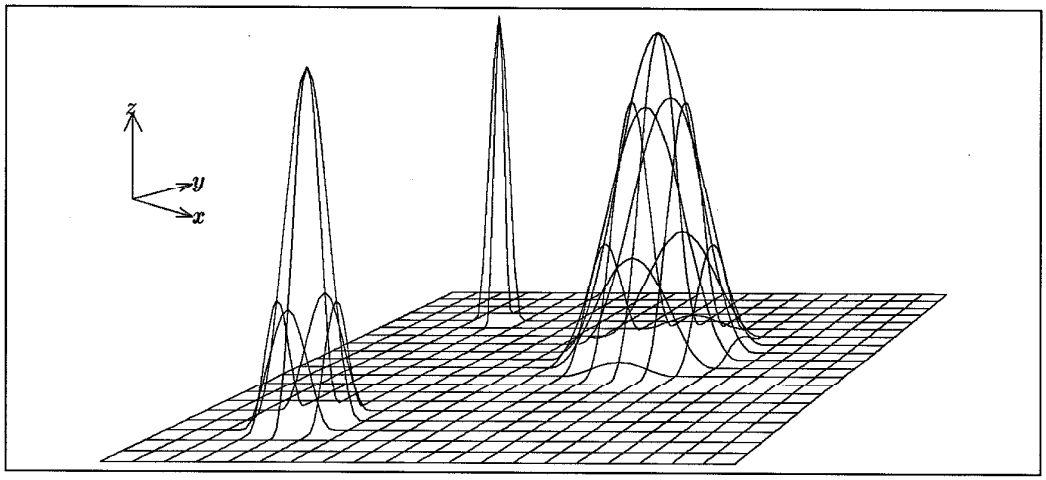

Figure 4.2: Parametric spike functions can be made arbitrarily sharp, so that their detection is extremely difficult. This makes collision detection arbitrarily difficult for parametric surfaces. We need some other information in order to guarantee the detection of the first intersection.

constraint on a parametric surface in order to guarantee that the first collision is detectable.

A simplistic approach for collision detection would be to position two surfaces at time t_1 and see if they intersect, and then move the surfaces to final positions at time t_2 and see if they intersect. We could then split the time difference and sample the two surfaces at time $(t_1+t_2)/2$, or some other time between t_1 and t_2. Recursing in this manner, we would sample the paths of the two surfaces. The problem with this technique for any finite number of samples is that we have no information about the positions of the surfaces between the sampling times. Without this information, we can never be sure that we have not missed an intersection. The problem is analogous to the spike problem of Figure 4.2.

To solve the collision-determination problem, we require a constraint on the maximum velocity of any point on the surface. If velocity is unconstrained, then the position of a surface may be discontinuous as a function of time, and the collision determination problem is insoluble (Appendix A.5). With knowledge of the maximum velocity of two surfaces, we can find the first collision of the surfaces.

4.1.3 Solution for Surfaces with Lipschitz Conditions

In Chapter 2, we applied a Lipschitz condition to parametric surfaces in order to create bounding volumes that completely contain the surface. Given a continuous

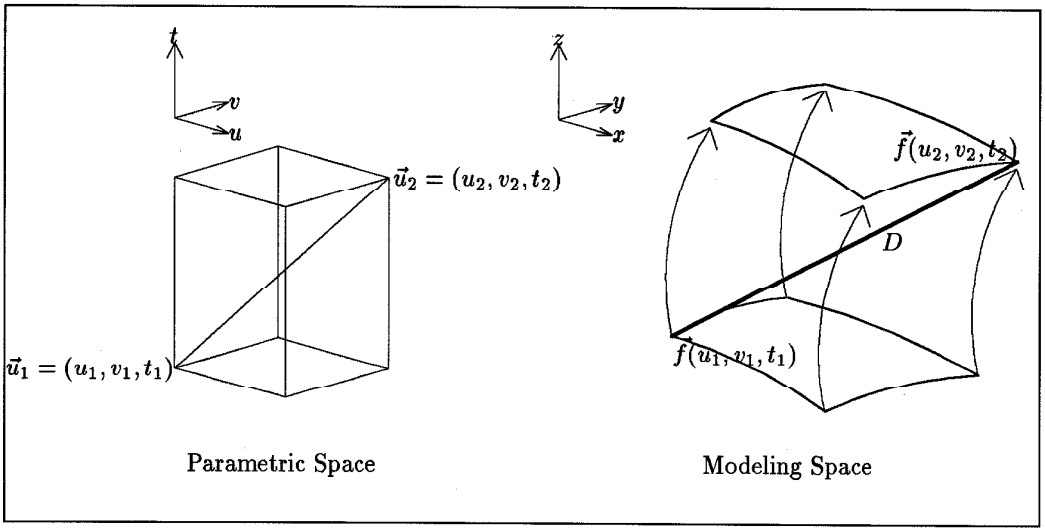

Figure 4.3: Graphical illustration of the Lipschitz inequality for parametric functions of three variables. We have $D \leq L \|\vec{u}_2 - \vec{u}_1\|$.

parametric surface $\vec{f}(\vec{u})$, the **Lipschitz condition** states that

$$\left\|\vec{f}(\vec{u}_2) - \vec{f}(\vec{u}_1)\right\| \leq L \|\vec{u}_2 - \vec{u}_1\| . \tag{4.2}$$

The Lipschitz condition is implied if the function $\vec{f}(\vec{u})$ has finite partial derivatives ([Lin and Segel 74, p. 58]). The Lipschitz constant L is a generalization of the derivative of $\vec{f}(\vec{u})$. We can also find Lipschitz constants for some surfaces that are not differentiable (see Section B.3). The Lipschitz condition on a surface is sufficient to create sets of bounding volumes that are guaranteed to bound the parametric surface.

It is possible to develop a similar constraint on the temporal aspects of the collision-determination problem. We can have a parametric surface $\vec{f}(\vec{u})$, $\vec{u} = (u, v, t)^T$, that moves as a function of time. We can construct a set of bounding volumes for the moving surface, in a manner analogous to the method for stationary surfaces. In this case, L sets an upper bound for the velocity of the parametric surface as well as for the other parametric derivatives. This inequality is depicted graphically in Figure 4.3.

Given parametric functions $\vec{f}_A(\vec{u})$ and $\vec{f}_B(\vec{u})$, along with their Lipschitz values L_A, and L_B, we will prove a method to determine the first ϵ-collision between two surfaces. Alternatively, we can confirm that two objects do not collide. In addition, we will generalize the notion of a Lipschitz constant so as to provide tighter bounding volumes for ϵ-collision computations.

The ϵ-collision algorithm is approximate, in that a distance tolerance ϵ determines the precision of the spatial computation. We will always find the first ϵ-collision if there is one. Given a tolerance ϵ, the algorithm will find points on each surface that lie within distance ϵ of each other. The algorithm has an advantage on machines with finite numerical precision, since it already accommodates uncertainty in the numerical values. If we had an exact algorithm, we would have to accommodate rounding errors by complicating the algorithm. In addition, using the ϵ-collision algorithm, we can compute whether two objects ever have a minimum separation that is less than a given value.

4.2 Previous Work

Previous techniques have used velocity and distance bounds for collision detection of rigid objects ([Culley and Kempf 86]). Upper bounds on velocity and lower bounds on distance can determine minimum time until the next collision between objects. There has been some work on determining lower bounds on distance for convex polygons and polyhedra ([Schwarz 81], [Cameron and Culley 86]), but relatively little has been done for parametric surfaces more complex than polynomial functions ([Bezier 74]).

The method for ϵ-collision determination presented in Section 4.5 applies to a wide class of parametric surfaces. The notion of an upper bound on velocity is generalized to parametric dimensions other than time. We can automatically find a lower bound to the separation distance between objects, given upper bounds to the parametric derivatives of the function. The derivative constraints enable us to sparsely sample a parametric function that deforms over time, and determine ϵ-collisions with other objects.

4.3 Finding Surface Intersections for Stationary Parametric Surfaces

First, we will describe a technique for determining if the surfaces of two stationary parametric functions intersect, if we are given Lipschitz constants for the surfaces. We construct surface networks for the two surfaces, and form a hierarchy of bounding volumes that completely bound each surface. Since we are not concerned with rendering cracks in the surface, it is not necessary to use restricted quadtrees to form the network.

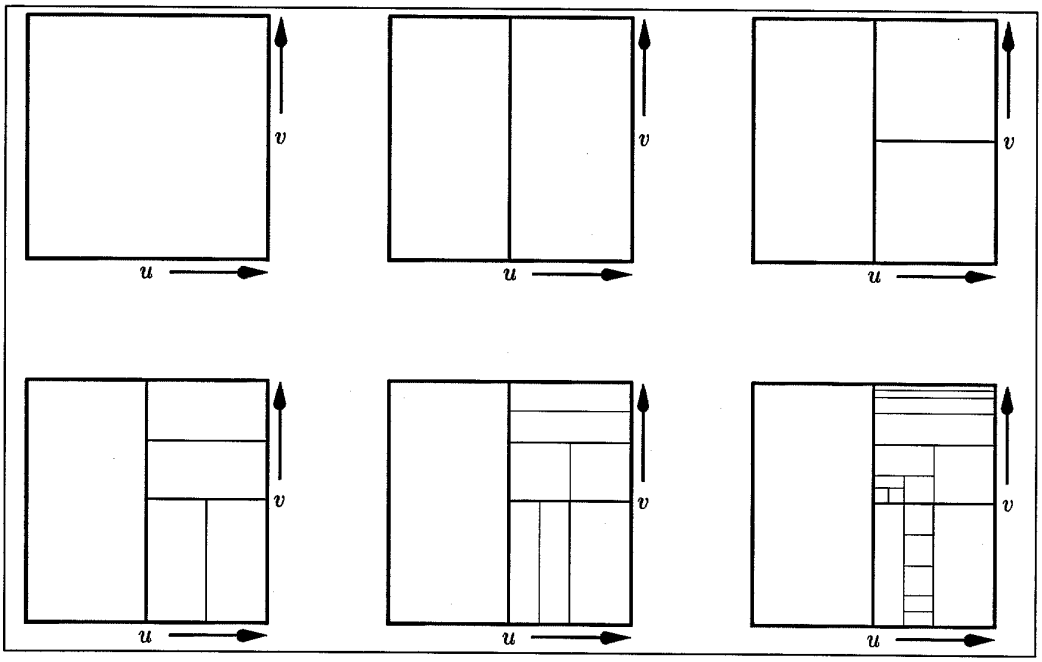

Figure 4.4: An example of successive subdivision levels of a 2-dimensional k-d tree spanning the unit square. The aspect ratio of the rectangles may be adjusted by powers of 2.

4.3.1 k-d Trees in Parametric Space

A variety of subdivision mechanisms are possible, including quadtrees of squares or bintrees of triangles. We choose to use an alternative to the quadtree, which generalizes to k dimensions, called the k-d tree (for k-dimensional binary search tree [Bentley and Friedman 79]). The reasons for this choice are that we need a method that extends easily to k dimensions, we are not concerned about rendering cracks, and we want good control over the aspect ratio of the parametric subregions. In the k-d tree method, k dimensional space is divided into k-dimensional boxes, using planes perpendicular to each of the parametric axes. Each subdivision level splits the k-dimensional box along one of the dimensions to form two descendent boxes.

The k-d tree offers more flexibility than the quadtree. We can control the aspect ratio of the parametric boxes by splitting the boxes several times in the same direction. Algorithms for the k-d tree adapt easily to arbitrary dimension since the data structure remains the same: a binary tree. We simply change the split direction as needed for the new dimensions. The total number of nodes may

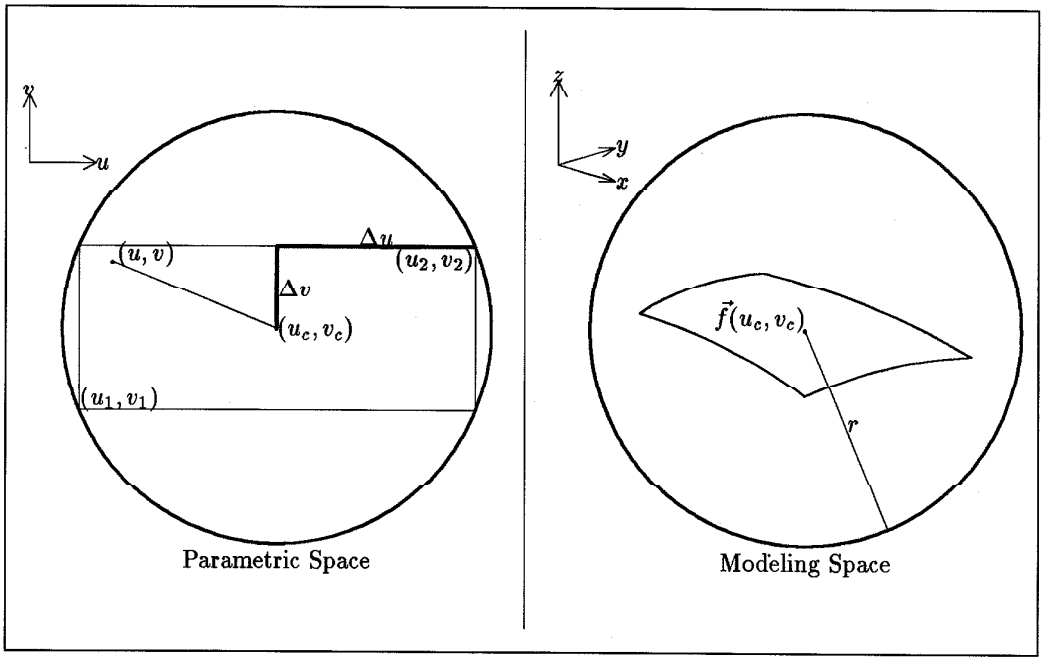

Figure 4.5: Computing the distance from a point (u, v) to the center of a rectangular subregion (u_c, v_c). A sphere of radius r bounds the parametric surface $\vec{f}(u, v)$ in the region $R : |u - u_c| \leq \Delta u$, $|v - v_c| \leq \Delta v$.

be smaller for the k-d tree, since only two new rectangles are generated for each subdivision level, as opposed to four for the quadtree. However, because of the lower fanout per level, the hierarchy of a k-d tree may require more subdivision levels than the quadtree.

4.3.2 Spherical Bounds in Modeling Space for Parametric Subregions

We first construct spherical bounding volumes for parametric rectangles and then extend the rectangles to higher dimensions. Given the L-function $\vec{f}(u, v)$ over a rectangular parametric region $R : |u - u_c| \leq \Delta u$, $|v - v_c| \leq \Delta v$, and the Lipschitz constant L for the surface, we have

$$\left\| \vec{f}(u, v) - \vec{f}(u_c, v_c) \right\| \leq L \left\| (u, v)^T - (u_c, v_c)^T \right\|. \tag{4.3}$$

We choose different norms for each side of the inequality:

$$\left\| \vec{f}(u, v) - \vec{f}(u_c, v_c) \right\|_2 \leq L \left\| (u, v)^T - (u_c, v_c)^T \right\|_1 \tag{4.4}$$

$$\left\|\vec{f}(u,v) - \vec{f}(u_c,v_c)\right\|_2 \leq L\left[|u-u_c| + |v-v_c|\right]. \tag{4.5}$$

We can generalize this equation with a separate constant for each parametric term:

$$\left\|\vec{f}(u,v) - \vec{f}(u_c,v_c)\right\|_2 \leq L_u|u-u_c| + L_v|v-v_c|, \tag{4.6}$$

where L_u and L_v are called the *rate constants* for the function $\vec{f}(u,v)$. From Eqn. 4.6, we can derive sufficient values for L_u and L_v, assuming that $\vec{f}$ is differentiable:

$$L_u \geq \max_R \left\|\frac{\partial \vec{f}}{\partial u}\right\|_2, \quad L_v \geq \max_R \left\|\frac{\partial \vec{f}}{\partial v}\right\|_2. \tag{4.7}$$

We take the maximum of the parametric derivatives over the parametric region R. The rate constants L_u and L_v are generalizations of the partial derivative, and are well defined for some non-differentiable surfaces (Appendix B.3).

We substitute the inequalities for region R to arrive at the **bounding sphere equation**:

$$\left\|\vec{f}(u,v) - \vec{f}(u_c,v_c)\right\|_2 \leq L_u\Delta u + L_v\Delta v, \tag{4.8}$$

for all points $(u,v)^T$ inside region R. The right side of Eqn. 4.8 represents the radius $r = L_u\Delta u + L_v\Delta v$ of a sphere in modeling space centered about $\vec{f}(u_c,v_c)$. Eqn. 4.8 states that any point of the parametric surface in the rectangular subregion R must lie within distance r from the center point $\vec{f}(u_c,v_c)$ of the subregion (Figure 4.5).

4.3.3 Intersection Computation

We can now compute the intersection of two parametric functions, to within a tolerance ϵ. We are given the parametric functions $\vec{f}_A(u,v), \vec{f}_B(u,v)$, the rate constants $L_{uA}, L_{uB}, L_{vA}, L_{vB}$, and a tolerance, ϵ, for the intersection determination. Our goal is to find a point $\vec{P}_A$ on surface $\vec{f}_A(u,v)$ and a point $\vec{P}_B$ on surface $\vec{f}_B(u,v)$, such that $\left\|\vec{P}_A - \vec{P}_B\right\|_2 \leq \epsilon$.

Without loss of generality, we can assume that both surfaces are defined over a unit parametric square given by $0 \leq u \leq 1$, and $0 \leq v \leq 1$. Starting with a single sample on each surface, $\vec{f}_A(0.5,0.5), \vec{f}_B(0.5,0.5)$, we determine the maximum radius of each subregion, using Eqn. 4.8. If the sum of the two radii are less than the distance between the two samples, then the two surfaces cannot possibly intersect, and we are done with the computation. On the other hand, if there is a potential overlap between the two surfaces, then we subdivide the larger region into two subregions, using a k-d tree, and compare each subregion with the other surface. The procedure recurses until we have checked all the subregions against each other, or until we have found a point on each surface in the same location to within ϵ.

RECURSIVE INTERSECTION OF TWO PARAMETRIC SURFACES

```
(defstruct node "Representation for a parametric surface region."
          parameters ; (u,v,t) coordinates of the center of the region.
          position   ; (x,y,z) coordinates of the center of the region.
          radius     ; Maximum radius of the region.
          child1     ; Pointer to first subregion of this region.
          child2)    ; Pointer to second subregion of this region.

(defun node-intersect (nodeA nodeB fnA fnB LA LB eps)
  "Compares nodes from two surfaces to see if they intersect."
  (cond
    ((> (separation-distance nodeA nodeB)
        (+ (node-radius nodeA)  ; If the nodes are separated by
           (node-radius nodeB))); more than the sum of their radii,
     nil)                       ; return a null intersection.
    ; Else if the distance is less than the tolerance, return the collision values.
    ((< (separation-distance nodeA nodeB) eps)
     (list 'collision-position   (node-position nodeA)
                                 (node-position nodeB))
           'tolerance            (separation-distance nodeA nodeB)
           'collision-parameters (node-parameters nodeA)
           'collision-parameters (node-parameters nodeB)))
    ; Else find the larger node radius, split it in two, and recurse.
    ((> (node-radius nodeA) (node-radius nodeB))
     (node-split nodeA fnA LA) ; nodeA is larger; split and recurse.
     (node-intersect (node-child1 nodeA) nodeB fnA fnB LA LB eps)
     (node-intersect (node-child2 nodeA) nodeB fnA fnB LA LB eps))
    (t ; Otherwise nodeB is larger; split and recurse.
     (node-split nodeB fnB LB)
     (node-intersect nodeA (node-child1 nodeB) fnA fnB LA LB eps)
     (node-intersect nodeA (node-child2 nodeB) fnA fnB LA LB eps))
    )))
```

Figure 4.6: An algorithm and data structure written in Common Lisp for recursively testing two parametric regions for overlap. The arguments **nodeA** and **nodeB** are the parametric regions to be intersected. The arguments **fnA** and **fnB** are pointers to the parametric function definitions. The arguments **LA** and **LB** are the rate constants. The value **eps** is the allowed tolerance of the result.

In effect, this algorithm implements the proximity subdivision criterion discussed in Section 2.5.4. In cases where the two surfaces are distant from each other, very little subdivision occurs. In regions where the two surfaces are in close proximity, many samples are taken in order to determine if the surfaces intersect, and if so, to determine where they intersect.

4.3.4 Intersection Algorithm in Common Lisp

Figure 4.6 shows the recursive algorithm, written in Common Lisp, that performs the intersection. The basic steps are:

1. Given a sample from each parametric surface, determine the bounding radius for each region.

2. Compare the sum of the two radii to the distance between the points in modeling space.

3. If there is an overlap, subdivide the larger region.

4. If the distance between the points is less than the tolerance, terminate.

5. Else recurse using the smaller region and the descendents of the larger region.

The data structure `node` represents a rectangular subregion of the parametric surface. The top-level instantiation of `node` represents the entire parametric surface over region R. The children of this node represent halves of the surface, and so on. `child1` and `child2` store pointers to descendent nodes that implement the k-d tree hierarchy. The fields `parameters` and `position` represent the domain and range, respectively, of each node. The field `radius` stores the maximum distance from any point in the region to the center point, based on the rate constants L_u and L_v for the function and on Eqn. 4.8.

The function `node-intersect` controls the subdivision of parametric regions to resolve potential intersections. The arguments `nodeA` and `nodeB` are parametric regions from surfaces A and B, respectively, that are to be compared against each other for potential intersection. The arguments `fnA` and `fnB` are pointers to the parametric functions that define surfaces $\vec{f}_A(u,v)$ and $\vec{f}_B(u,v)$. These functions take parametric coordinates and return surface positions in modeling space. The values `LA` and `LB` are the rate values L_u and L_v for surfaces $\vec{f}_A(u,v)$ and $\vec{f}_B(u,v)$. The rate values L_u and L_v may be constants or may be pointers to functions (see the next subsection). The functions take nodes as arguments, and return L values that are valid for the argument region. Finally, `eps` represents the allowed

tolerance of the intersection values. The intersection computation recurses until the desired tolerance is reached.

The function `node-intersect` calls several other functions. One of these functions, `separation-distance`, computes the distance in modeling space between the center of `nodeA` and the center of `nodeB`. The function `node-split` takes a node and subdivides it along one of the parametric axes, instantiating two new nodes, `child1` and `child2`. When a node is split, the parametric axis for the split is determined by the largest of $(L_u\Delta u, L_v\Delta v)$. This way we always divide the dimension that contributes most to the radius r of the region.

4.3.5 Extension to Variable L Values over a Surface

In the preceding analysis, we have treated the L_i as constants. We can generalize Eqn. 4.8 by considering the rate values L_u and L_v to be a function of the parametric region R:

$$\left\|\vec{f}(u,v) - \vec{f}(u_c,v_c)\right\|_2 \leq L_u(R)\Delta u + L_v(R)\Delta v. \tag{4.9}$$

Two L values can form tighter bounds on the parametric function $\vec{f}(u,v)$ than one L value can. While a constant value of L requires the evaluation of the maximum partial derivative over the entire surface, a function $L(R)$ can evaluate the maximum partial derivative over smaller regions. The inequalities for L_u and L_v in Eqn. 4.7 are evaluated for each new subregion of R that is created. This approach may lead to substantial improvements in efficiency for some types of surfaces with dramatic changes in parametric derivatives.

4.3.6 Sphere Example

As an example, we derive the L values for a unit parametric sphere. The spherical function is given by

$$\vec{f}(u,v) = \begin{pmatrix} \cos(2\pi u)\sin(\pi v) \\ \sin(2\pi u)\sin(\pi v) \\ -\cos(\pi v) \end{pmatrix}. \tag{4.10}$$

Substituting into Eqn. 4.7, we obtain rate constants L valid over the entire surface:

$$L_u = 2\pi, \quad L_v = \pi. \tag{4.11}$$

If, instead, we generate functions for L over a parametric region $R : |u-u_c| \leq \Delta u$, $|v-v_c| \leq \Delta v$, we can improve the fit for L_u:

$$L_u(R) = 2\pi\cos(\pi\max(0, |v_c - 1/2| - \Delta v)), \quad L_v(R) = \pi. \tag{4.12}$$

A derivation of the result for $L_u(R)$ appears in Appendix B.1. Near the polar regions of the parametric sphere, the bounding volumes become considerably smaller for the functional definition of $L_u(R)$.

4.4 Bounding Volumes for Moving Parametric Surfaces

We are now in a position to extend the bounding volume results of Section 4.3 to the problem of parametric surfaces moving as a function of time. The method presented here is general enough to determine collisions of flexible objects whose shape changes as a function of time. We still use k-d trees to perform the parametric subdivision, but now we are dealing with parametric rectangular prisms, rather than with rectangles. We must traverse the parametric volumes of two surfaces to verify that they do not collide.

4.4.1 Bounding Spheres Derived from the Rate Condition

A straightforward way to generate bounding volumes for parametric functions of three variables is to use a modification of Eqn. 4.8:

$$\left\| \vec{f}(u,v,t) - \vec{f}(u_c,v_c,t_c) \right\|_2 \leq L_u(R)\,|u-u_c| + L_v(R)\,|v-v_c| + L_t(R)\,|t-t_c|\,, \tag{4.13}$$

where $t_c = (t_1 + t_2)/2$, and t_1 and t_2 are the temporal limits of the parametric volume of interest. We obtain a sample at the center of the parametric volume and then compute a radius in modeling space for a sphere in which the surface element must remain during the time interval specified. We redefine region R to be

$$R = \left\{ (u,v,t)^T : \ |u-u_c| \leq \Delta u, \ \ |v-v_c| \leq \Delta v, \ \ |t-t_c| \leq \Delta t \ \right\}. \tag{4.14}$$

Initially, we set the region R to be a unit cube

$$0 \leq u \leq 1, \ \ 0 \leq v \leq 1, \ \ 0 \leq t \leq 1, \tag{4.15}$$

that corresponds to the domain of $\vec{f}_A(u,v,t)$ and $\vec{f}_B(u,v,t)$. Starting with a single sample in the center of each parametric cube, $\vec{f}_A(0.5,0.5,0.5)$ and $\vec{f}_B(0.5,0.5,0.5)$, we determine the radius of each bounding sphere according to

$$r = L_u \Delta u + L_v \Delta v + L_t \Delta t. \tag{4.16}$$

If the two spheres overlap, we recursively subdivide the regions until they no longer overlap, or until the minimum tolerance ϵ is reached. If the overlapping spheres are smaller than ϵ, we report a collision.

4.4.2 Bounding Volumes Based on the Jacobian of the Parametric Function

It is possible to improve on the bounding volume results of the previous section. The goal is to minimize the size of the bounding volumes, thus reducing the average number of interference computations.

We start with the original definition of the Lipschitz condition for parametric functions ([Gear 71]):

$$\left\| \vec{f}(\vec{u}) - \vec{f}(\vec{u}_c) \right\| \leq L \left\| \vec{u} - \vec{u}_c \right\| . \tag{4.17}$$

We choose an L_1 norm for the right side of Eqn. 4.17, and we apply the condition to each component of $\vec{f}$ separately:

$$\begin{aligned} |x(u,v,t) - x(u_c,v_c,t_c)| &\leq L_x \left(|u-u_c| + |v-v_c| + |t-t_c| \right), \\ |y(u,v,t) \quad y(u_c,v_c,t_c)| &\leq L_y \left(|u-u_c| + |v-v_c| + |t-t_c| \right), \\ |z(u,v,t) - z(u_c,v_c,t_c)| &\leq L_z \left(|u-u_c| + |v-v_c| + |t-t_c| \right), \end{aligned} \tag{4.18}$$

for some suitable values of L_i. We distribute the values L_i and rename them to arrive at a more general inequality:

$$\begin{aligned} |x(u,v,t) - x(u_c,v_c,t_c)| &\leq M_{xu} |u-u_c| + M_{xv} |v-v_c| + M_{xt} |t-t_c| , \\ |y(u,v,t) - y(u_c,v_c,t_c)| &\leq M_{yu} |u-u_c| + M_{yv} |v-v_c| + M_{yt} |t-t_c| , \\ |z(u,v,t) - z(u_c,v_c,t_c)| &\leq M_{zu} |u-u_c| + M_{zv} |v-v_c| + M_{zt} |t-t_c| . \end{aligned} \tag{4.19}$$

We can solve for each M_{ij} by choosing appropriate values of (u,v,t). We illustrate with M_{xu}:

$$|x(u,v,t) - x(u_c,v,t)| \leq M_{xu} |u-u_c| , \tag{4.20}$$

or

$$\left| \frac{x(u,v,t) - x(u_c,v,t)}{u-u_c} \right| \leq M_{xu}, \quad u \neq u_c. \tag{4.21}$$

Assuming that $x(u,v,t)$ is differentiable, a sufficient value of M_{xu} is

$$M_{xu} \equiv \max_R \left| \frac{\partial x(u,v,t)}{\partial u} \right| . \tag{4.22}$$

The parametric derivative is a lower bound on the maximum value of M_{xu} over the region R. In general, a sufficient value of the **rate matrix** is:

$$\mathbf{M} \equiv \begin{pmatrix} \max\limits_R \left| \frac{\partial x}{\partial u} \right| & \max\limits_R \left| \frac{\partial x}{\partial v} \right| & \max\limits_R \left| \frac{\partial x}{\partial t} \right| \\ \max\limits_R \left| \frac{\partial y}{\partial u} \right| & \max\limits_R \left| \frac{\partial y}{\partial v} \right| & \max\limits_R \left| \frac{\partial y}{\partial t} \right| \\ \max\limits_R \left| \frac{\partial z}{\partial u} \right| & \max\limits_R \left| \frac{\partial z}{\partial v} \right| & \max\limits_R \left| \frac{\partial z}{\partial t} \right| \end{pmatrix} . \tag{4.23}$$

Just as the Lipschitz value L is a generalization of the derivative, so the rate matrix $\mathbf{M}$ is a generalization of the Jacobian matrix for parametric vector functions of several variables. The matrix $\mathbf{M}$ consists of upper bounds on all the parametric derivatives of all the components of vector function $\vec{f}$. $\mathbf{M}$ is just a generalization of the Jacobian $\mathbf{J}$ of the parametric function $\vec{f}(u,v,t)$ [Lin and Segel 74, p. 355]:

$$\mathbf{J}(u,v,t) \equiv \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial u} & \frac{\partial z}{\partial v} & \frac{\partial z}{\partial t} \end{pmatrix} \tag{4.24}$$

We define parametric coordinates (u_c, v_c, t_c) at the center of region R, and modeling space coordinates $x_c \equiv x(u_c, v_c, t_c)$, $y_c \equiv y(u_c, v_c, t_c)$, and $z_c \equiv z(u_c, v_c, t_c)$. We also define $\Delta u \equiv |u_2 - u_c|$, $\Delta v \equiv |v_2 - v_c|$, and $\Delta t \equiv |t_2 - t_c|$. The distance from the center point (u_c, v_c, t_c) to any other point in R cannot be greater than the half-widths of each dimension of the bounding box (see Figure 4.5):

$$\begin{aligned} |u - u_c| &\leq \Delta u, \\ |v - v_c| &\leq \Delta v, \\ |t - t_c| &\leq \Delta t. \end{aligned} \tag{4.25}$$

Substituting into Eqn. 4.19, we have the **rate condition**:

$$\begin{aligned} |x(u,v,t) - x(u_c,v_c,t_c)| &\leq M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ |y(u,v,t) - y(u_c,v_c,t_c)| &\leq M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ |z(u,v,t) - z(u_c,v_c,t_c)| &\leq M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t. \end{aligned} \tag{4.26}$$

We define the **bounding box radii** to be

$$\begin{aligned} \Delta x &\equiv M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ \Delta y &\equiv M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ \Delta z &\equiv M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t. \end{aligned} \tag{4.27}$$

Now we can construct a bounding volume from the bounding box radii. We form a rectangular prism that is aligned with the x, y, and z axes, centered about parametric point (u_c, v_c, t_c), and centered about modeling coordinates (x_c, y_c, z_c). Combining Eqn. 4.27 with Eqn. 4.26, we get the **bounding box inequality**:

$$\begin{aligned} |x - x_c| &\leq \Delta x \\ |y - y_c| &\leq \Delta y \\ |z - z_c| &\leq \Delta z. \end{aligned} \tag{4.28}$$

Such a rectangular region is called an *isothetic rectangle,* a rectangle whose sides are parallel to coordinate axes [Lee and Preparata 84]. The set of points satisfying Eqn. 4.28 form a bounding box containing the parametric region. We now have an efficient bounding box useful for computing collisions between moving parametric surfaces. We are free to compute the Jacobian maxima over the entire surface, thereby computing with a single-valued constant matrix across the surface. Alternatively, we may compute the Jacobians over subregions in order to tailor the bounding volumes more closely to particular variations in the surface. These boxes frequently produce tighter bounds on the parametric functions than do the spheres of Section 4.4.

4.5 Algorithm for Collision Determination

This section deals with the computation of collisions from the bounding box information of the previous section. We are given the parametric functions $\vec{f}_A(u,v,t)$ and $\vec{f}_B(u,v,t)$. We are also given a function that returns a value greater than or equal to the maximum of the absolute value of each element of the Jacobian matrix for the parametric function over a rectangular range of parameters, which is called the rate matrix **M**. We require that

$$\begin{array}{lll} M_{xu} \geq \max_R \left|\frac{\partial x}{\partial u}\right| & M_{xv} \geq \max_R \left|\frac{\partial x}{\partial v}\right| & M_{xt} \geq \max_R \left|\frac{\partial x}{\partial t}\right| \\ M_{yu} \geq \max_R \left|\frac{\partial y}{\partial u}\right| & M_{yv} \geq \max_R \left|\frac{\partial y}{\partial v}\right| & M_{yt} \geq \max_R \left|\frac{\partial y}{\partial t}\right| \\ M_{zu} \geq \max_R \left|\frac{\partial z}{\partial u}\right| & M_{zv} \geq \max_R \left|\frac{\partial z}{\partial v}\right| & M_{zt} \geq \max_R \left|\frac{\partial z}{\partial t}\right|. \end{array} \tag{4.29}$$

The task is to compute whether two objects collide, as determined by the loss of separation of the two parametric surfaces. We assume initially that the two objects are disjoint. We are given a threshold distance tolerance, ϵ, below which we should report the loss of separation, and the time at which separation was lost. We should also report the parametric locations of the contact point. The method will guarantee that the first collision between the surfaces occurs within the time interval returned as the collision time.

4.5.1 Collision Algorithm Approach

The basic method is similar to that of Figure 4.6, but there are some important differences. We need not only to detect a collision between surfaces, but also to

detect the first collision between surfaces. This implies that we should traverse the nodes of the k-d trees in forward-time order rather than the depth-first recursive order of Figure 4.6. In particular, we schedule pairs of nodes (one from each surface) to be compared against each other according to the earliest time that the two surfaces could possibly collide. This is determined from the minima of the time bounds of the parametric subregions. We compute the intersection of the time intervals of the two nodes. The two parametric regions cannot collide until they both have come into existence. So the maximum of the two starting times represents the earliest possible collision time. In other words, given the time interval $t_A \pm \Delta t_A$ of node A, and the time interval $t_B \pm \Delta t_B$ of node B, we sort the node pairs according to the earliest possible intersection time t_{min}:

$$t_{\text{min}} = \max(t_A - \Delta t_A, t_B - \Delta t_B). \tag{4.30}$$

We maintain a heap data structure [Knuth 69] of pairs of nodes to be compared, sorted in ascending order, using t_{min} as the sort key. Each record stores a pointer to a node from the first parametric surface, and a pointer to a node from the second surface against which the first surface is evaluated. We successively pop node pairs off the heap for comparison, in ascending order, according to t_{min}. If the node comparison generates new node pairs to be evaluated, they are pushed onto the heap, and are sorted as necessary to ensure forward-time traversal of the parametric space. This method guarantees that we will find the first collision between the surfaces.

4.5.2 Common Lisp Implementation

Figure 4.7 shows an algorithm written in Common Lisp for computing the collision between two parametric surfaces. The `node` data structure is similar to the node representation for intersection detection of parametric surfaces. The `surface-collision` function computes an ϵ-sphere that contains points from both surfaces, or else confirms that the two surfaces do not collide.

Several functions are called by the `surface-collision` function. The function `setup-initial-node` computes an initial node for the surface at parametric location $(0.5, 0.5, 0.5)$. The function `schedule-node-pair` takes a pair of nodes, sees if they overlap in time and in space, computes the t_{min} value, and pushes them onto the heap to be scheduled for evaluation. The operation `heap-pop` pops a pair of nodes off the heap for evaluation. The function `collision-information` returns the collision parameters if an ϵ-collision took place. Finally, the function `node-split` subdivides a node into two smaller nodes along the parametric dimension with the greatest contribution to the bounding box size.

COLLISION DETERMINATION FOR TWO PARAMETRIC SURFACES

```
(defstruct node "Representation for a parametric surface region."
          parameters ; (u,v,t) coordinates.
          position   ; (x,y,z) coordinates in modeling space.
          radii      ; Bounding box radii in x, y, and z.
          child1     ; Pointer to first subregion of this region.
          child2)    ; Pointer to second subregion of this region.

(defun surface-collision (fnA fnB MA MB tolerance)
  "Compares nodes from two surfaces to see if they collide."
  (let ((heap (make-heap 'compare-node-pairs)))
    ; Push the initial node pair on the heap for evaluation.
    (schedule-node-pair (setup-initial-node fnA MA)
                        (setup-initial-node fnB MB) heap)
    (catch 'heap-empty ; If the heap is ever empty, terminate.
      (loop ; Get the next pair off the heap, and test them.
       (multiple-value-bind (nodeA nodeB) (heap-pop heap)
         (cond ; If the tolerance is reached, terminate.
           ((enough-precision nodeA nodeB tolerance)
            (return (collision-information nodeA nodeB)))
           ((> (max (node-radii nodeA)) ; Split the largest node.
               (max (node-radii nodeB))) ; Put pairs on the heap.
            (node-split nodeA fnA MA)
            (schedule-node-pair (node-child1 nodeA) nodeB heap)
            (schedule-node-pair (node-child2 nodeA) nodeB heap))
           (t ; If nodeB is bigger than nodeA, split nodeB.
            (node-split nodeB fnB MB)
            (schedule-node-pair nodeA (node-child1 nodeB) heap)
            (schedule-node-pair nodeA (node-child2 nodeB) heap))
           ))))))
```

Figure 4.7: An algorithm and data structure for testing for the collision of two parametric surfaces moving as a function of time. The arguments `fnA` and `fnB` are pointers to the parametric function definitions. The arguments `MA` and `MB` are rate matrices for the surfaces A and B. The value `tolerance` is the allowed separation uncertainty of any collisions.

4.5.3 Termination Condition

The function **enough-precision** determines whether an ϵ-sphere contains both surfaces. For termination, we compute the smallest isothetic rectangle that contains the two bounding boxes. If the largest dimension of the isothetic rectangle is smaller than the separation tolerance, we report the loss of separation of the two surfaces, down to the tolerance specified. Expressed mathematically, for bounding boxes $(x_A, y_A, z_A) \pm (\Delta x_A, \Delta y_A, \Delta z_A)$ and $(x_B, y_B, z_B) \pm (\Delta x_B, \Delta y_B, \Delta z_B)$, and tolerance ϵ, we require

$$\max \begin{pmatrix} |x_A - x_B| + \Delta x_A + \Delta x_B, \\ |y_A - y_B| + \Delta y_A + \Delta y_B, \\ |z_A - z_B| + \Delta z_A + \Delta z_B \end{pmatrix} \leq \epsilon. \tag{4.31}$$

See Appendix A.1 for the definition of outer diameter d_o for a pair of boxes. This criterion causes the recursion to terminate for any ϵ-collisions between surface A and surface B (see Appendix A.4).

The collision algorithm has an important property: Parametric surfaces that are far apart will be shown not to collide, using a single sample from each surface. This computation is extremely fast, making it computationally trivial to reject collisions between distant objects. The closer the approach of the two surfaces, the longer it takes to determine that the two objects do not collide, because more of the bounding boxes overlap.

4.5.4 Complexity Analysis for Colliding Spheres

We would expect that as the separation distance decreases between two spheres, the number of bounding box comparisons should increase. In particular, if the separation distance drops by a factor of two, we will have to create bounding boxes twice as small to confirm that the surfaces do not intersect. For the parametric k-d tree hierarchy, every halving of the separation distance requires a constant number of additional subdivision levels. Assuming that CPU time should be proportional to the number of parametric nodes created, the CPU time t should scale as

$$t \propto \log_2 ((r + S)/S), \tag{4.32}$$

where r is the radius of each sphere, and S is the separation distance between spheres. The argument to the logarithm is always greater than 1, for positive r and S, so that the CPU time t is always positive. Let us see if this principle is confirmed experimentally.

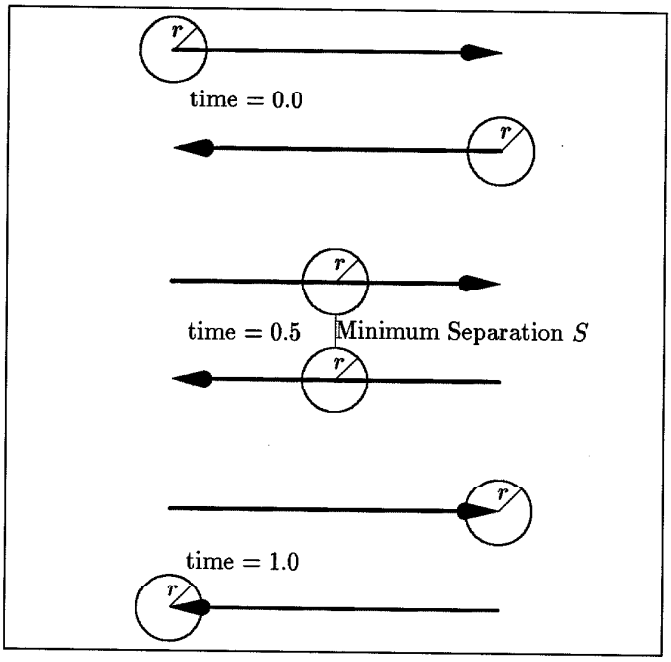

Figure 4.8: Collision experiment between two spheres of radius r. The CPU time for collision determination is measured as a function of separation distance S.

4.5.5 Experimental Results for Colliding Spheres

As an illustration of the relationship between computation time and separation distance S, Figure 4.8 shows the setup for a series of collision experiments, using two objects that pass each other at successively shorter distances. The total computation time is a function of the minimum separation distance between the two objects. The graph in Figure 4.9 shows the computation time for each run in the series. For an object of radius r and minimum separation distance $2r$, we require only a few samples to be taken from each surface. As the minimum separation distance decreases, we notice an increase in CPU time proportional to the negative logarithm of the separation distance.

4.5.6 Experimental Results for Other Objects

As a demonstration of results for surfaces more complicated than polynomials or quadrics, the collision method is demonstrated for two parametric spike illustrated

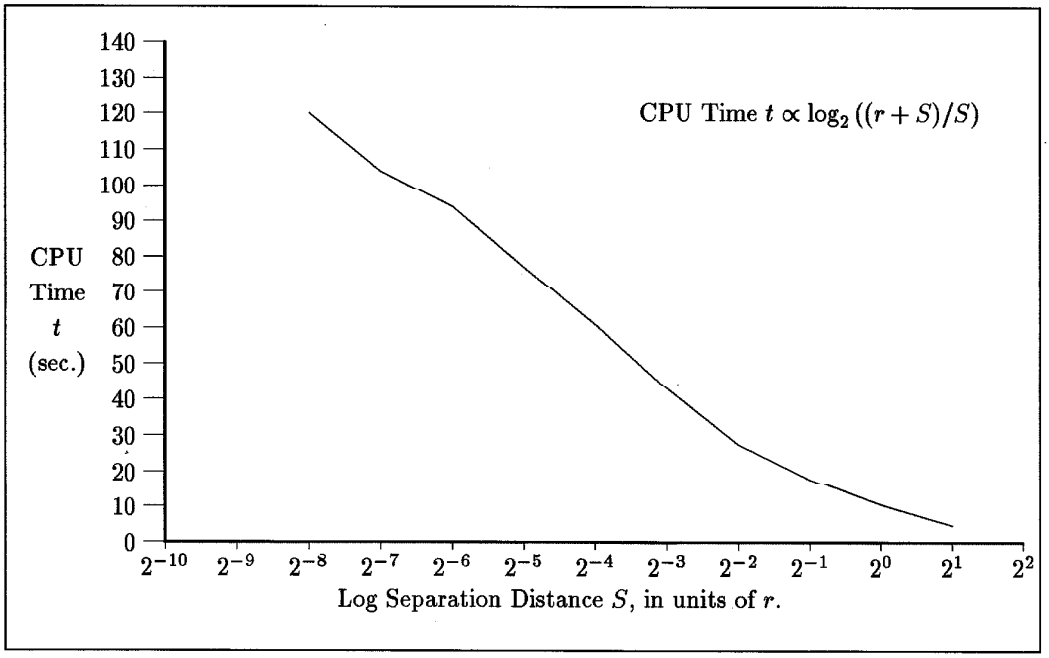

Figure 4.9: Example of typical CPU time as a function of $\log S$, where S is the minimum separation between the two objects of Figure 4.8.

in Figure 4.10. The parametric equation for the spike function is

$$\vec{f}(u,v) = \begin{pmatrix} r(u,v)\cos(2\pi u)\sin(\pi v) \\ r(u,v)\sin(2\pi u)\sin(\pi v) \\ -r(u,v)\cos(\pi v) \end{pmatrix}, \tag{4.33}$$

where the radius is given by

$$r(u,v) = r_0 + r_1 \sum_{i=0}^{i<n} e^{-((u-u_i)^2+(v-v_i)^2)/w_0^2}. \tag{4.34}$$

The value n is the number of spikes on the sphere, (u_i, v_i) is the parametric location of the i-th spike, and w_0 determines the radius of the spikes.

Without knowing something about the parametric derivatives of the spike function, it would be very difficult to solve the collision problem for two moving spike functions. As it is, we are able to construct a set of bounding volumes as the computation requires, in order to verify the paths of the two objects.

Figure 4.11 shows the results of a collision computation between two spherical spike functions. In Figure 4.10, we see two spherical spike functions approaching

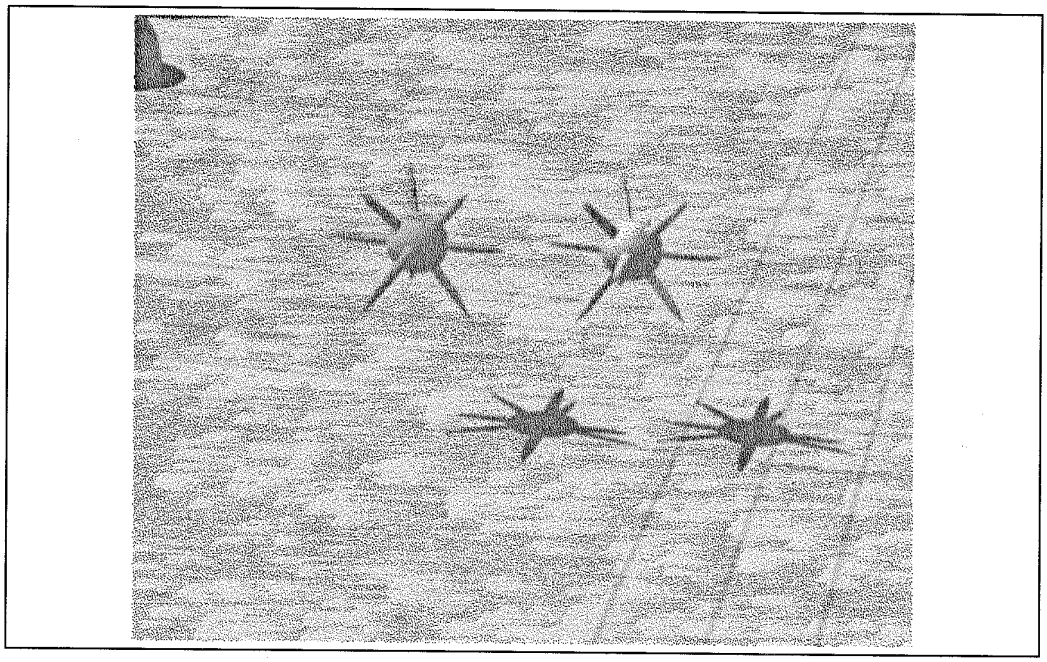

Figure 4.10: A pair of spherical spike functions before a collision.

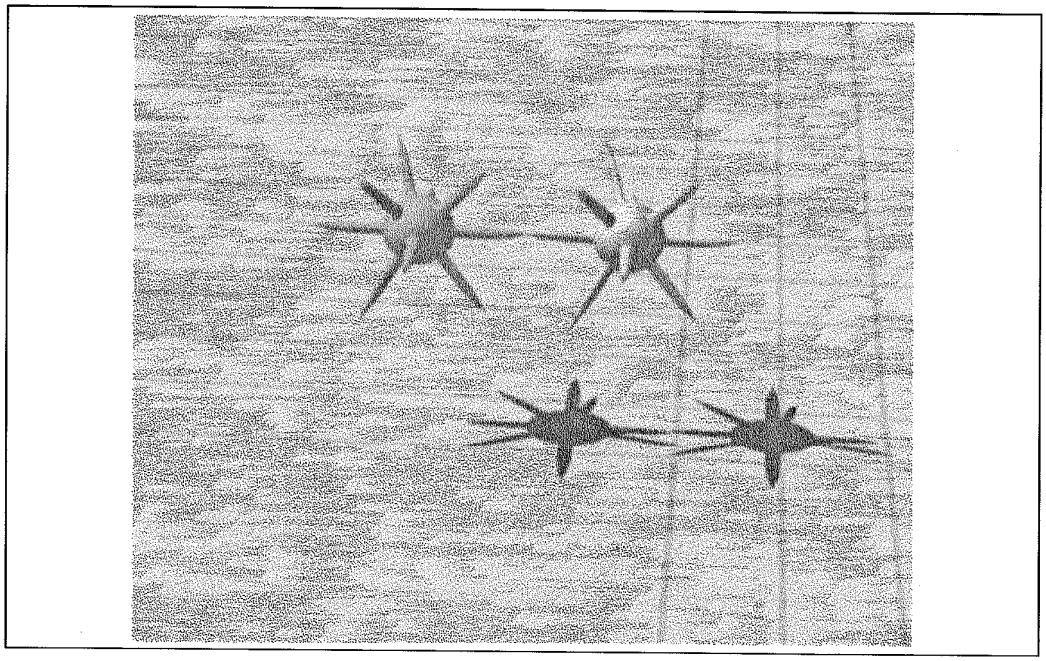

Figure 4.11: A pair of spherical spike functions during a collision.

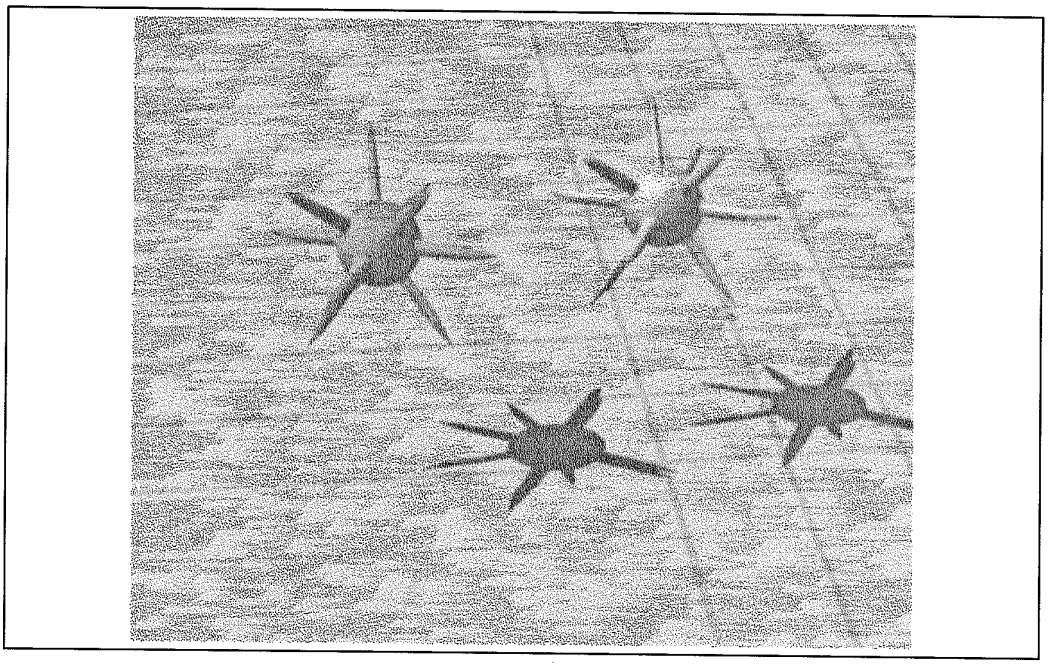

Figure 4.12: A pair of spherical spike functions after a collision.

each other. In Figure 4.11, the algorithm computes a collision between two of the spikes, causing the two objects to recoil, as shown in Figure 4.12. This collision computation would have been very difficult to solve without knowing the rate matrices for the spherical spike function. With this information, we can solve difficult collision problems, using a straightforward application of the collision algorithm of Figure 4.7.

4.6 Determining Constraints on the Jacobian of a Parametric Function

For the collision technique to be useful, we need to determine constraints on the Jacobian of the parametric functions. We may have only a global maximum of any component of the Jacobian, or we may be able to determine an analytic equation for each component of the Jacobian, and for the Jacobian maximum over any region.

4.6.1 Global Maximum of all the Components

The simplest approach is to compute the maximum of any component of the Jacobian over the entire surface, and then to set each entry of the rate matrix M equal to the maximum value. This does not provide particularly tight bounds on the parametric surface, but is sufficient to compute collisions.

4.6.2 Global Maximum for each Parametric Variable

An intermediate solution is to have separate maxima for velocity and for spatial extent. It is common for the time derivatives, such as $\partial x/\partial t$, to have separate scaling from the spatial parametric derivatives, such as $\partial x/\partial u$ and $\partial x/\partial v$. It is also common for the u and v derivatives to have separate scalings. If we define

$$\begin{aligned} w_u &\equiv \max_R \left(\left|\frac{\partial x}{\partial u}\right|, \left|\frac{\partial y}{\partial u}\right|, \left|\frac{\partial z}{\partial u}\right| \right), \\ w_v &\equiv \max_R \left(\left|\frac{\partial x}{\partial v}\right|, \left|\frac{\partial y}{\partial v}\right|, \left|\frac{\partial z}{\partial v}\right| \right), \\ w_t &= \max_R \left(\left|\frac{\partial x}{\partial t}\right|, \left|\frac{\partial y}{\partial t}\right|, \left|\frac{\partial z}{\partial t}\right| \right), \end{aligned} \tag{4.35}$$

where R is the domain of the entire parametric function, then the following matrix constrains the Jacobian of the parametric surface:

$$\mathbf{M}(R) = \begin{pmatrix} w_u & w_v & w_t \\ w_u & w_v & w_t \\ w_u & w_v & w_t \end{pmatrix}. \tag{4.36}$$

Each column of $\mathbf{M}$ has a separate entry: either a constant for the whole surface, or a function of subregion R. We obtain a set of bounding volumes tighter than with the single constant approach, but without the analytical complexity of separate entries for each component of $\mathbf{M}$.

4.6.3 Global Maximum of each Component

We may carry the process one step further and simply compute maxima for each component of the Jacobian over the entire surface. This approach is a direct application of Eqn. 4.23 to the parametric functions of interest. We simply compute an upper bound on each component of the Jacobian for each surface, and assemble these to form a rate matrix of constants for each parametric function:

$$M_{ij} \geq \max_R |J_{ij}|, \tag{4.37}$$

where R represents the domain of the entire surface over the complete time interval of the motion. This method avoids a function call to the rate-matrix function each time the bounding radii of subregions are computed, since the rate matrix is constant for all subregions. Yet this approach has the flexibility that each component of the Jacobian has a distinct upper bound that may be significantly different from one component to another.

4.6.4 Local Maximum of each Component

Perhaps the most general and flexible way to compute constraints on the Jacobian matrix is to create a special function that computes maxima of the parametric derivatives of each parametric function. If we can find an analytical solution to the Jacobian of the parametric function, and a function for the maximum of every component in the Jacobian over an arbitrary parametric range, then we can produce very tight bounds around a surface. In some cases, we may be able to provide an exact analytic solution to the **M** function. At other times, we may need to use approximation rules to the various components. We must satisfy only the condition that

$$M_{ij} \geq \max_R |J_{ij}| . \tag{4.38}$$

In this case, R may be any subregion of the parametric domain of the function.

4.6.5 Identities for Computing the Maxima of Functions over a Domain

We can use several identities to compute the maxima of a function. Given functions $\vec{f}(R)$ and $\vec{g}(R)$ defined over a parametric domain R, we can use the following identities to reduce the complexity of computation of maxima over regions:

$$\max_R |\vec{f}(R) + \vec{g}(R)| \leq \max_R |\vec{f}(R)| + \max_R |\vec{g}(R)|, \tag{4.39}$$

$$\max_R |\vec{f}(R) - \vec{g}(R)| \leq \max_R |\vec{f}(R)| + \max_R |\vec{g}(R)|, \tag{4.40}$$

$$\max_R |\vec{f}(R)\vec{g}(R)| \leq \max_R |\vec{f}(R)| \max_R |\vec{g}(R)|, \tag{4.41}$$

$$\max_R |\vec{f}(R)/\vec{g}(R)| \leq \frac{\max_R |\vec{f}(R)|}{\min_R |\vec{g}(R)|}, \tag{4.42}$$

for all $\vec{f}(R)$ and $\vec{g}(R)$.

The maximum of a function over a two-dimensional region must occur either within the interior of the region, or along the edges of the region, or at the corners of the region. We can use results from differential geometry to compute local maxima in these three cases, and to obtain a global maximum for the region.

4.6.6 Matrix Computation for a Parametric Sphere

For the case of a unit sphere moving with non-constant velocity $\vec{s} = (s_x, s_y, s_z)$, we have

$$\mathrm{M}(R) = \begin{pmatrix} 2\pi \max\limits_R |\sin 2\pi u| \max\limits_R |\sin \pi v| & \pi \max\limits_R |\cos 2\pi u| \max\limits_R |\cos \pi v| & \max\limits_R |s_x| \\ 2\pi \max\limits_R |\cos 2\pi u| \max\limits_R |\sin \pi v| & \pi \max\limits_R |\sin 2\pi u| \max\limits_R |\cos \pi v| & \max\limits_R |s_y| \\ 0 & \pi \max\limits_R |\sin \pi v| & \max\limits_R |s_z| \end{pmatrix}.$$

The derivation of this result may be found in Appendix B.2.

Chapter 5

Applications of Adaptive Sampling with Surface Networks

This chapter discusses several applications for surface networks. First we consider the problem of reducing a texture map to a set of linearly shaded polygons that approximate the texture map. The technique can help to produce shaded images that contain texture maps, using polygon renderers. Then we consider applications of the theory for collision determination developed in Chapter 4. Potential applications to robotics and aviation are discussed.

5.1 Triangulation of Texture Maps

5.1.1 Conversion of Texture Maps into Polygon Tilings

In this section, the sampling techniques developed so far are applied to the problem of converting texture maps into sets of polygons that approximate surface patterns and coloration. A texture map is defined typically as an array of pixels in a rectangle that represents a two-dimensional image. Many rendering systems can accommodate polygons with colored vertices ([Swanson and Thayer 86]), but cannot accommodate texture maps as an array of colors positioned across a surface. The technique presented here permits the adaptive transformation of a texture map into a set of polygons that can be rendered using conventional polygon renderers.

Another potential benefit is to compress the coloring information into a more concise form. Usually a texture map is stored as an array of pixels in a map at some given resolution. Adaptive techniques promise to reduce the data required to approximate surface coloration. Because the tolerance is adjustable, coarse coloration may be provided for previewing, and detailed texturing may be performed for final images.

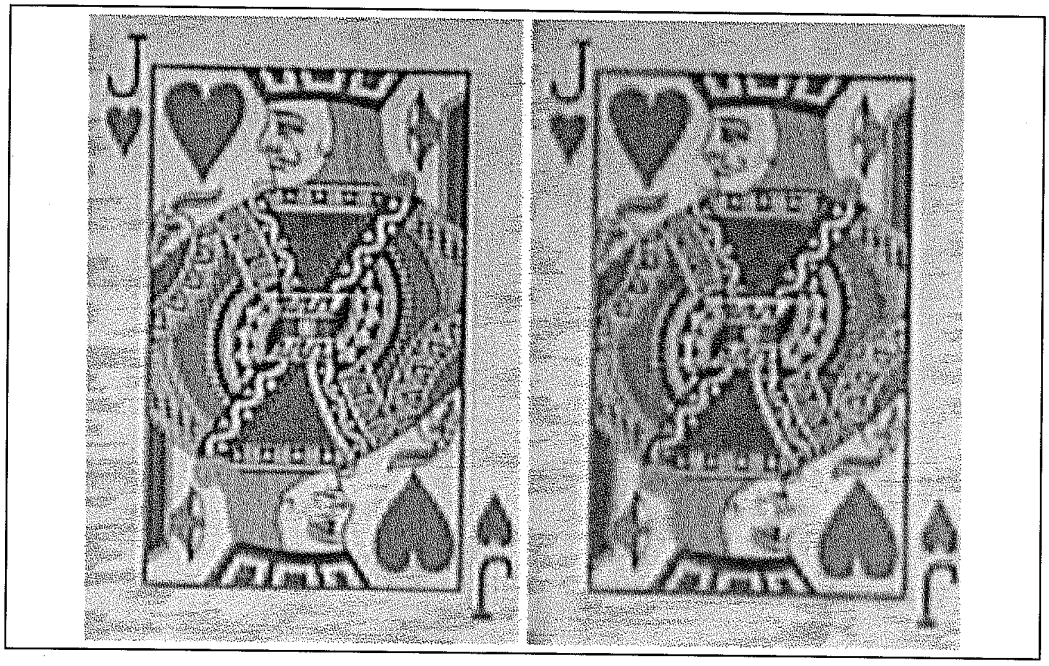

Figure 5.1: A texture map of a playing card. The left image shows the original texture map of the jack of hearts. The right image shows a triangulation of the same texture map using bintrees of right triangles.

A third benefit is the improvement in rendering speed for animation of textures. With current polygon rendering systems, it is possible to render over 20,000 polygons in less than a second ([Swanson and Thayer 86]). Current texture mapping techniques frequently require much longer times to filter and to render the same texture.

The polygonal tiling of a textured image must be able to approximate the shading discontinuities that occur across the boundaries of objects. It should also represent concisely the smooth shading variations that are due to surface curvature. The technique we propose satisfies both of these requirements. All of the distinct features of the texture map should be accurately represented.

In this paper we discuss the application of surface networks to the problem of texture map triangulation. The basic conversion method is fast and simple, at the cost of lossy compression of the data. The ability of bintrees of right triangles to find most of the features in texture maps that have high spatial derivatives speaks for the robustness of the sampling technique. (Compare the texture map with the triangulation in Figure 5.1).

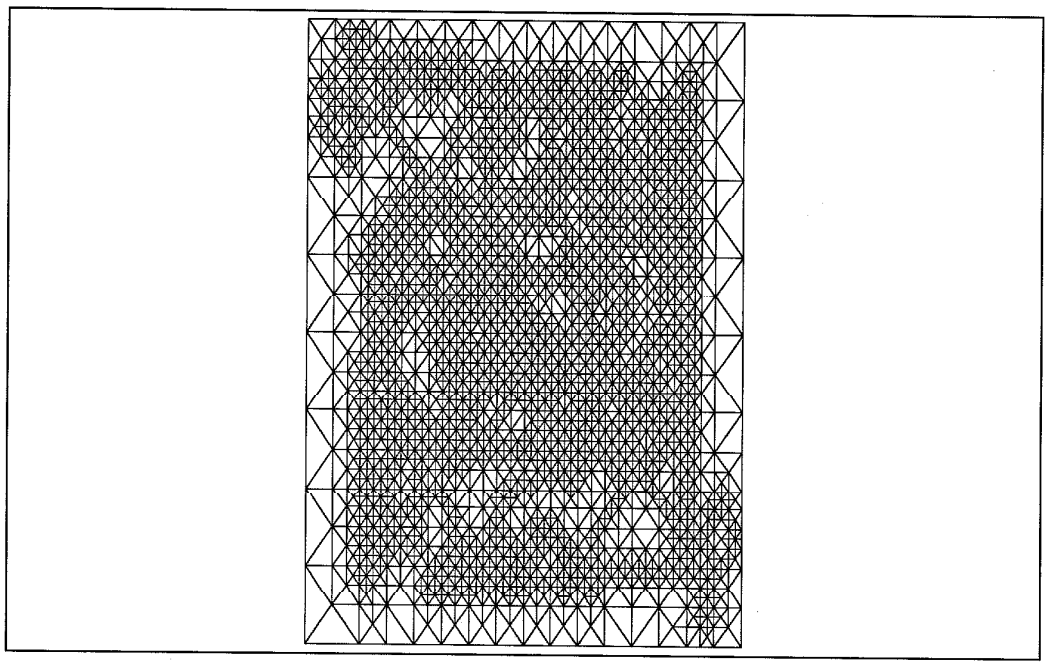

Figure 5.2: A triangular surface network of the jack of hearts.

5.1.2 Previous Work

Several representations have been used previously to compute images with texture maps. Summed area tables [Crow 84] store at each pixel location, not the color of the pixel, but the sum T of the intensity I at each pixel in a rectangle from one corner of the image to that pixel:

$$T_{mn} \equiv \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij}. \tag{5.1}$$

It is possible to determine the average color of any rectangular region across the surface of an image with one addition and two subtractions. This technique is good for low-pass filtering of texture maps, useful when a texture map covers a small apparent region in an image. But the technique does not address methods of rendering a texture, using a polygon renderer.

Another texture-mapping technique involves the use of mip maps, in which a single texture is represented in a map at a multitude of resolutions ([Williams 83]). A particular representation is selected based on the desired spatial frequency components of the map. Interpolation is performed between maps to provide a smooth transition from one resolution to another as the image scale varies. This technique

can be slow on machines optimized for polygon rendering.

A third approach is discussed in [Besl and Jain 88], in which the local Gaussian and mean curvature of the intensity values of the texture map are computed at each pixel. The regions are grown by merging pixels of similar curvature of the intensity surface into polynomial approximations of second-, third-, or fourth-order polynomial functions. The technique segments an image into smooth pieces separated by discontinuities. It is also necessary to convert the polynomial functions into sets of polygons for rendering on a polygon renderer.

We propose to convert a texture map into sets of polygons, using a binary triangular subdivision mechanism that recursively subdivides the parametric plane. The subdivision criterion is based on the color difference between the vertices of a triangle. If the difference is too large, the triangle is subdivided until some limiting resolution is reached. Triangles may be either flat-shaded or Gouraud-shaded to produce the image. The object is to tile the parametric plane with a triangular subdivision such that the plane is fully covered and no overlapping occurs. A triangular subdivision will guarantee that no cracks appear in the surface (Section 2.4). Low-pass filtering is accomplished by limiting the minimum size of each polygon in the tiling. Since each polygon is linearly shaded across its surface, this limits the spatial frequencies of the texture map.

5.1.3 Subdivision Mechanism Using Right Triangular Subdivisions of the Plane

A trivial method of converting a texture map into a set of polygons is to make a polygon for each pixel in the texture map, and to render each polygon individually. The problem with this method is that we may end up with a million polygons. We would like to reduce the number of polygons by a factor of ten or more without seriously degrading the quality of the map.

We use bintrees of right triangles to control the subdivision process. The subdivision mechanism for bintrees of right triangles is shown in Section 3.5. For simplicity, we transform the image onto a unit parametric square for adaptive sampling. The unit square is initially spanned by two right triangles, as shown in Figure 5.3. The subdivision mechanism always splits the triangles along their longest edge. If two right triangles share a common hypotenuse, then subdivision may occur immediately (Figure 3.6); otherwise, subdivision must propagate to larger triangles until we can find a pair that do share the hypotenuse. A triangle with a hypotenuse along the edge of the sampling region can always be immediately subdivided, since it has no neighbor. It is possible to prove that the recursion will always terminate, since it must proceed from small triangles to larger ones, and there is a limit to the maximum size of the triangles. In this way, we create a

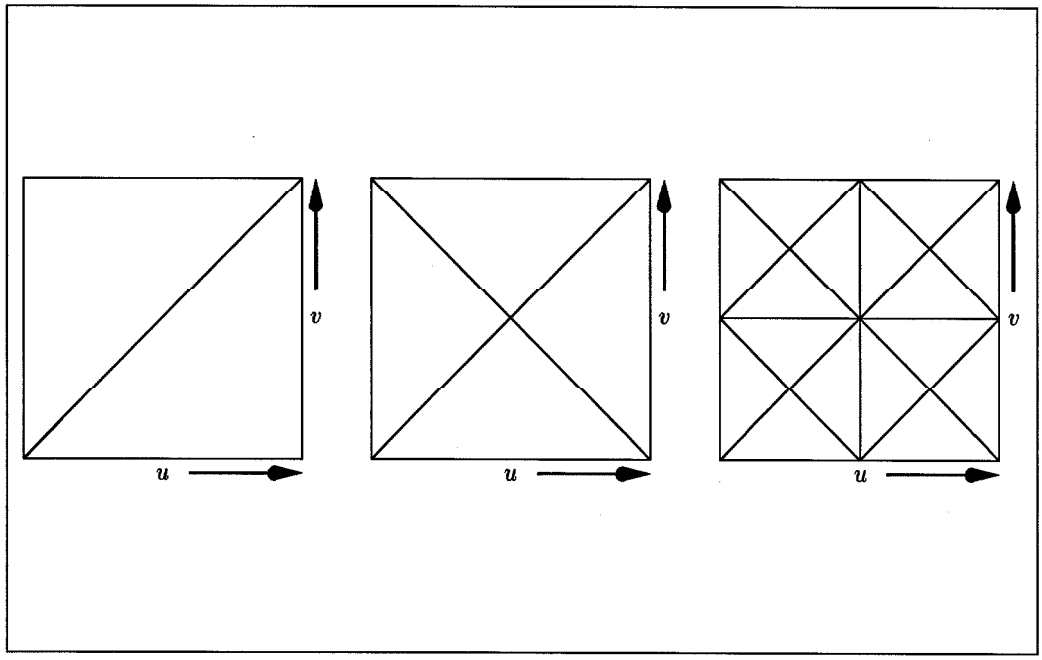

Figure 5.3: Subdivision mechanism for textures using bintrees of right triangles. The parametric region on the left shows the initial configuration of triangles on the unit parametric square. The middle figure demonstrates the splitting process, where subdivision is allowed only by splitting the hypotenuse of the triangles. The right figure shows the network after the third uniform subdivision.

triangular subdivision in which vertices always touch vertices, never edges, thereby maintaining C_0 continuity across the intensity surface.

5.1.4 Subdivision Criterion

The subdivision criterion used is to subdivide a triangle whenever the color difference between the vertices exceeds some specified value. The criterion forces additional sampling near features with high contrast, with lower sampling rates in regions of low contrast. The nature of the subdivision is such that there is a smooth transition from small triangles to large ones, which improves the robustness of the algorithm by exploring in the vicinity of regions of high contrast.

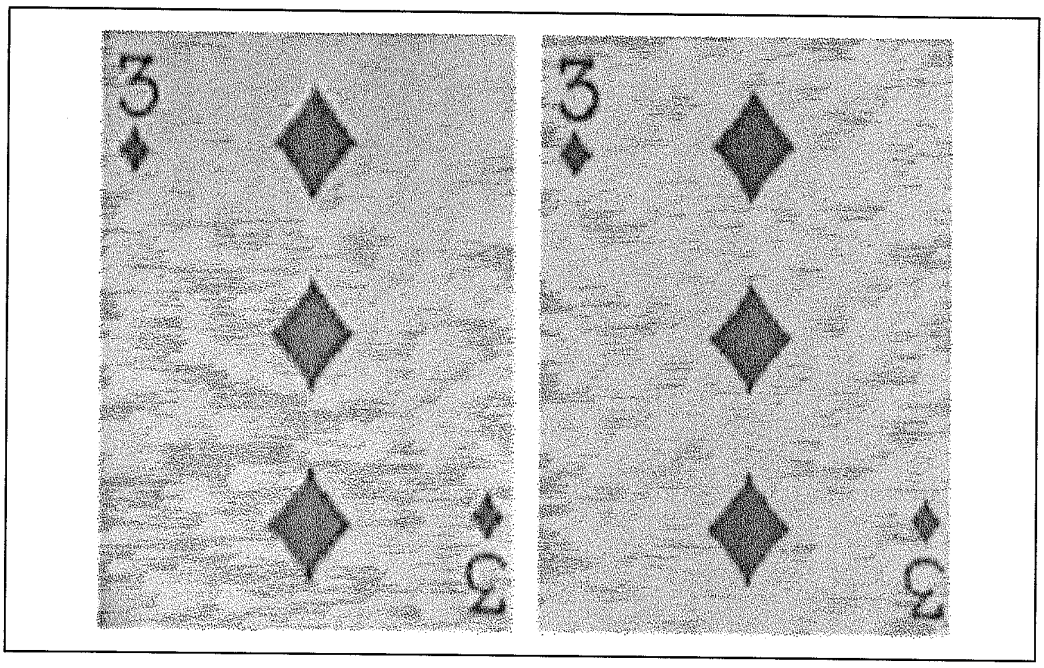

Figure 5.4: The image on the left shows a triangulation of a texture map. On the right is the same triangulation after merging the triangles into larger polygons. The texture on the right has three times fewer polygons, and takes half as much time to render as the triangular version on the left.

5.1.5 Merging Step

Once we have created a set of triangles representing the texture map, we perform a merging operation to further optimize the number of polygons. We take advantage of the fact that for some textures, such as the numerical playing cards of Figure 5.6 and Figure 5.4, much of the texture is covered by regions of constant shade that are punctuated by smaller regions of other constant shades. Gouraud shading may be rotation-invariant only for triangles, but flat shading will work fine for large, concave polygons. The algorithm is to merge these large regions of constant shade into a few large polygons that will render much more quickly than the large set of triangles. Figure 5.5 shows a set of polygons for a texture before and after the merging process.

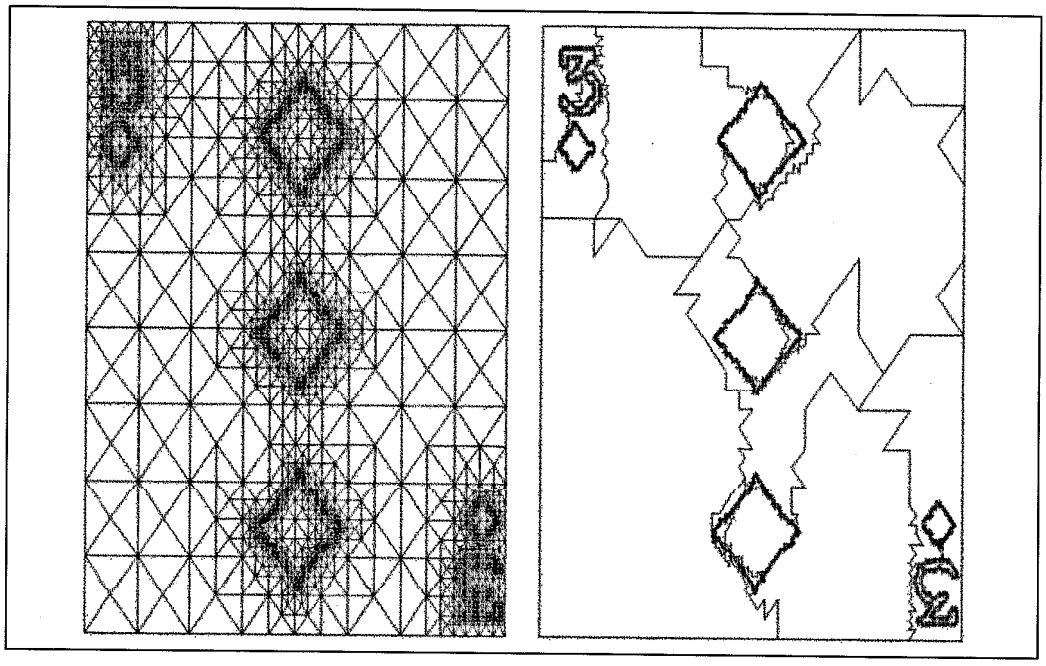

Figure 5.5: The image on the left shows the outlines of the triangles for the three of diamonds. On the right we see the outlines of the merged polygons. In regions of constant shading, the merging algorithm greatly reduces the number of polygons required.

5.1.6 Polygons Represented as Circular Lists

We choose a representation for polygons as circular lists of edges around the perimeter of the polygon to facilitate the merging operation. [Knuth 69] describes a circular list, or ring, as a list whose last element points back to the first element. The merging of two adjacent polygons is done by splicing together the two rings representing the polygons, and removing adjacent edges that are identical (a null circuit). The merging process is analogous to computing a line integral around a region by summing the line integrals of each subregion making up the region (Figure 5.9). Figure 5.7 shows a triangle represented as a ring of edges A, B, and C. Figure 5.8 shows a pair of triangles made up of edges (A, B, C) and (C, D, E). In order to merge these two polygons, we find a common edge between the two polygons, in this case edge C, and merge the two rings of edges at that point, forming (A, B, C, C, D, E). An invariant under line integrals and rings is to remove opposite edge traversals that are adjacent to each other: $(A, B, C, C, D, E) = (A, B, D, E)$. Once we have simplified the expression, we have a new list of edges that make up the perimeter of the merged polygon. In this way, it is possible to build up

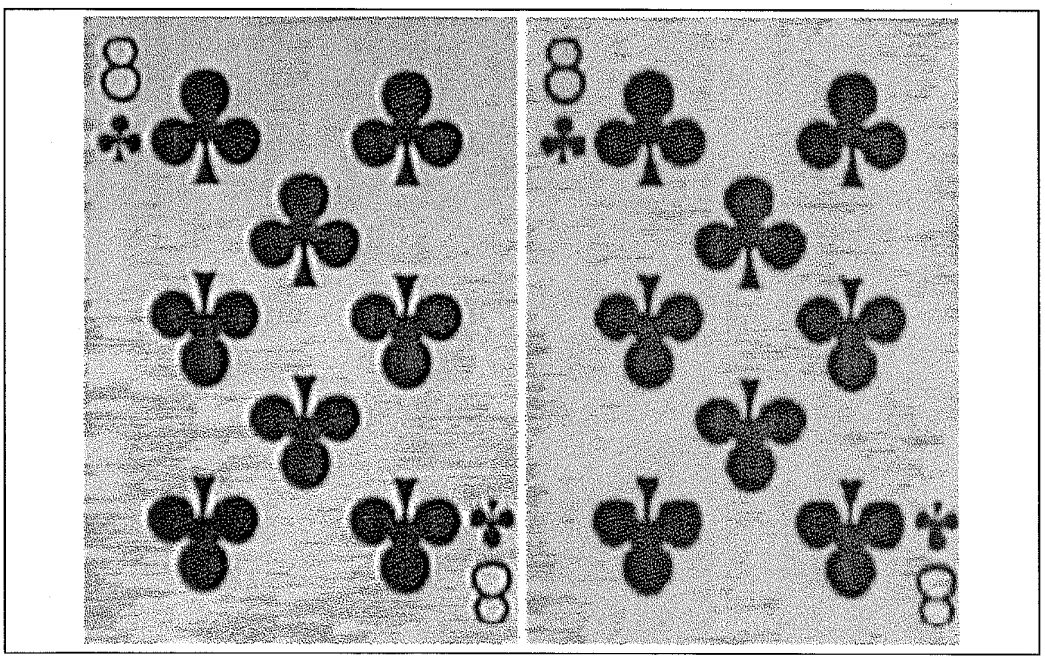

Figure 5.6: The left image shows a texture map of the eight of clubs. The right image shows a triangulation of the same texture map.

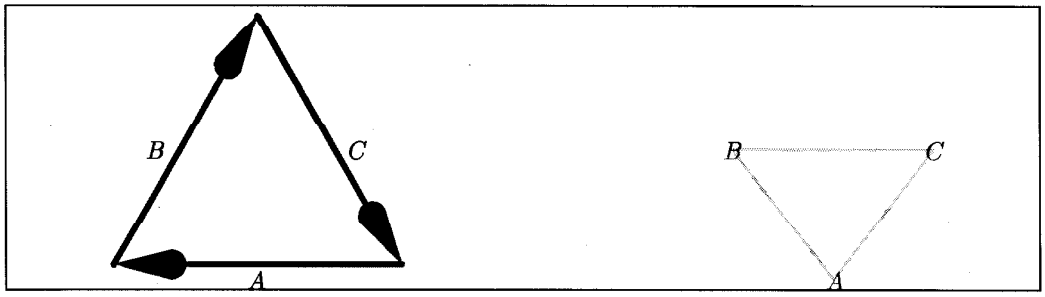

Figure 5.7: A triangle can be represented as a circular list of edges around the perimeter of the polygon. This structure is useful for polygon-merging operations. We refer to the circular list of edges as (A, B, C).

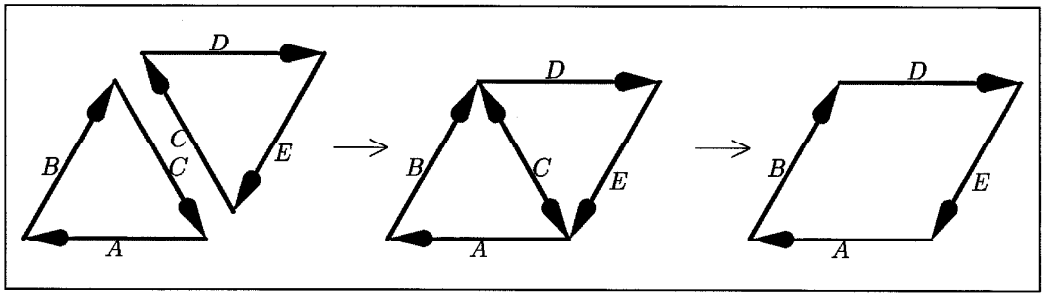

Figure 5.8: Illustration of the merging process for two triangles. Triangles with edges ABC and CDE, on the left, are represented as circular lists (A, B, C) and (C, D, E). In the middle figure, the triangle lists are merged into a single structure representing the path (A, B, C, C, D, E). As a final step, the redundant elements of the path are removed, in the figure at right, leaving path (A, B, D, E).

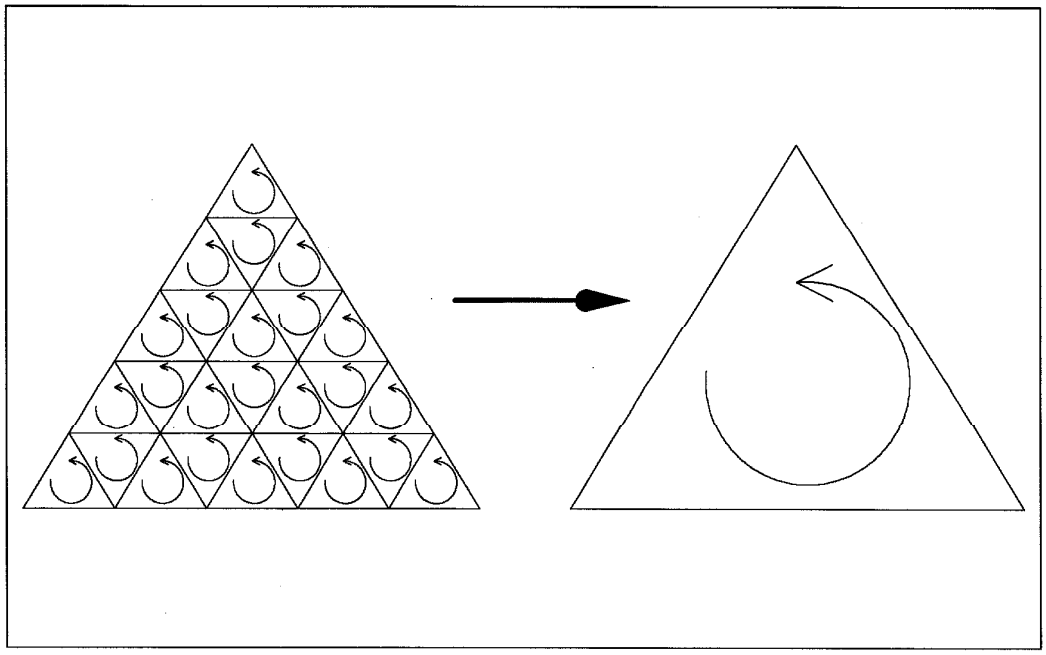

Figure 5.9: Illustration that the line integral of a perimeter is equal to the sum of line integrals of small finite elements.

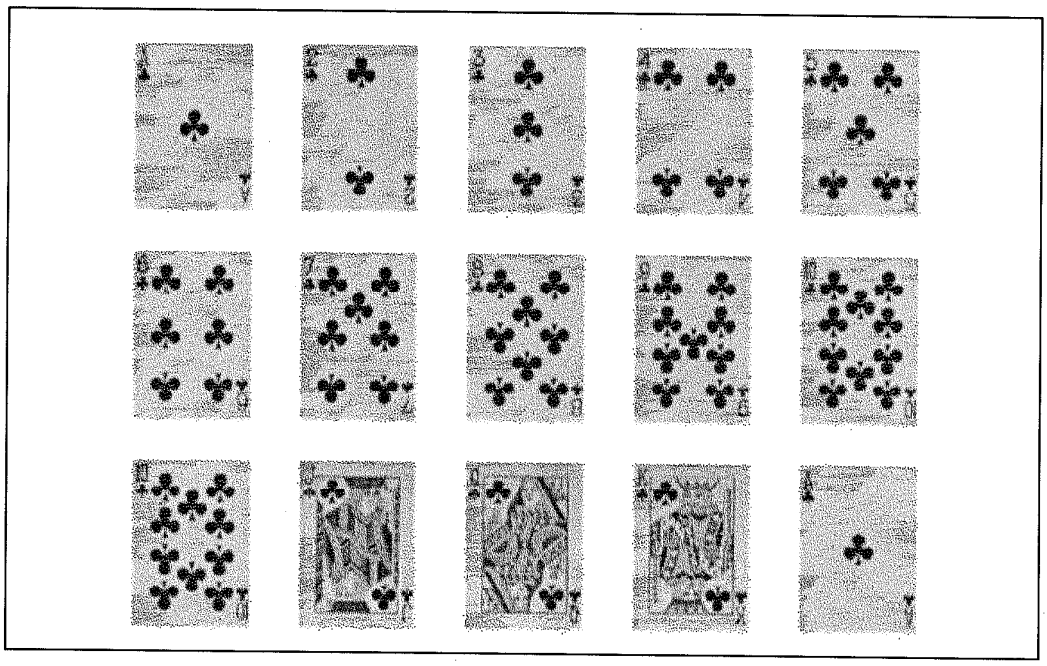

Figure 5.10: Triangulations of the suit of clubs.

homogeneous polygons with hundreds of vertices from triangular surface networks. This has the effect of speeding up the rendering of texture maps such as playing cards. Figure 5.4 shows two versions of a playing card, with and without polygon merging in the flat-shaded regions. The texture without triangle merging uses 4981 triangles, while the texture with merging uses 1527 triangles. The rendering time on an HP9000/350 SRX system is 1.5 seconds for triangles only, and 0.7 seconds for triangles with merging in the flat-shaded regions.

5.1.7 Imaging Results and Applications

The texture conversion algorithms have been tested on a set of texture maps representing the 52 playing cards in a standard deck, plus a texture map of the back of the playing cards. Figure 5.1 shows a texture map and its triangulation. Figure 5.2 shows a surface network of the texture map made up of right triangles.

We have converted a complete suit of cards into polygons to illustrate the technique (Figure 5.10). The number of polygons, and the rendering time, are roughly proportional to the perimeter of the distinct regions in the image. The face cards require significantly greater time, since the summed perimeter is much greater for these patterns.

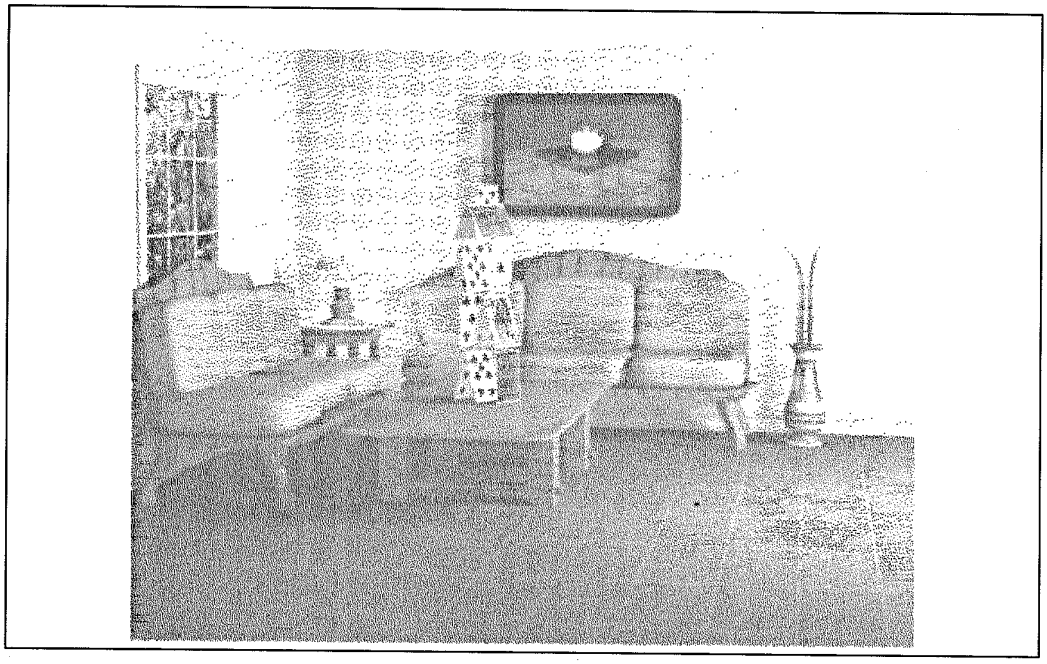

Figure 5.11: Frame from an animation using triangulated texture maps.

These images of playing cards were used for an animation illustrating constrained, rigid-body dynamics ([Barzel and Barr 88]). A frame from the animation is shown in Figure 5.11. The animation is roughly one minute long, animated at a rate of 30 frames per second. Rendering speed is critical for animation purposes, since typical animation sequences require thousands of frames. The animation would have been extremely slow on the available workstations without the capability of converting textures into polygons. Adaptive sampling together with polygon merging resulted in significantly faster animation of the dynamic constraints.

5.2 Potential ϵ-Collision Applications

In this section we apply the results of Chapter 4 to time-dependent problems such as collisions. We examine several potential applications of the collision theory to robotics and air-traffic control. These examples are meant to illustrate the general fields in which the ϵ-collision theory may apply.

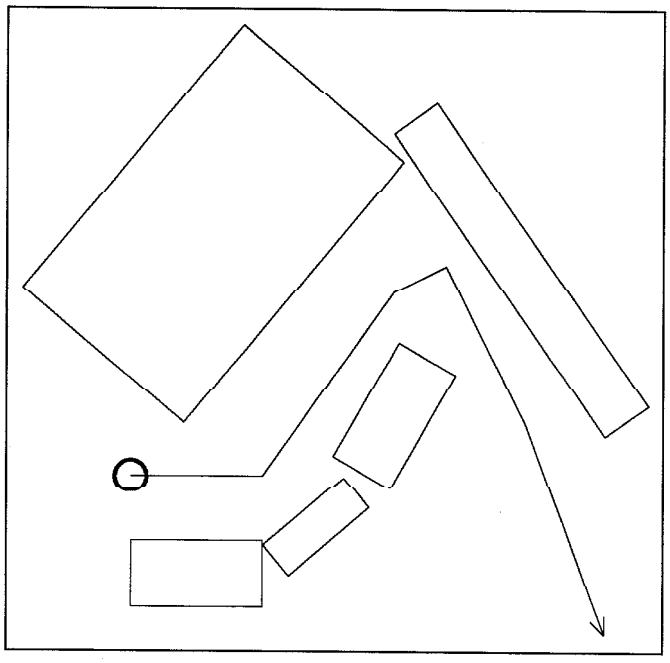

Figure 5.12: Path of a robot navigating a cluttered environment. We must determine if the robot will collide with any objects.

5.2.1 Robotic Path Verification

The rate matrix method for collision detection can potentially verify the path of a robotic system through a complex environment. The problem may be set up as follows: Given a parametric description of a robot, its motion as a function of time, and a surrounding dynamic environment, determine if the robot collides with its environment. The robot and the environment can be made up of a large set of parametric surfaces.

The solution to this problem is useful for robotic path planning, using a generate-and-test paradigm. Trial paths are generated and then tested using the collision-determination procedure described in Chapter 4. If the trial path results in a collision, the path is modified to avoid the collision and is retested. In this manner, it may be possible to do automatic path planning for robots.

Since the method using Jacobians is quite general, it is not particularly difficult to accommodate a moving environment as opposed to a static one. For simplicity we illustrate a spherical robot navigating a static array of boxes, as shown in Figure 5.12.

The technique used is to start with a single sample from each parametric func-

tion, and to take additional samples as necessary to verify the path of the robot. The average computation time is only weakly proportional to the number of objects, since most surfaces are far away from the robot at any given point in time. Only when the robot and an obstacle are near each other do we have to do substantial subsampling in order to verify the robot path.

5.2.2 Collision Prediction for Aircraft

An important problem in metropolitan areas is the increasing congestion of air traffic near major airports. Recent articles have talked of the possibility of gridlock in the nation's air transportation system ([New York Times, June 19, 1988]) because of aging airport facilities and the major expansion of air-traffic over the last decade. Air transportation currently has a market of $57 billion per year, or 1.5 percent of the gross national product of the United States. Airlines also provide 92 percent of the public transportation between cities in the United States. Techniques for air-traffic control have not kept pace with the growth in aviation.

One of the major problems with increased air traffic is the coordination of arriving and departing traffic in major metropolitan areas. Controllers have the major responsibility of preventing collisions between aircraft near major airports, yet increased traffic is making it difficult to use manual methods of air-traffic control.

It is important to address potential solutions to the problem of recognizing potential collisions between aircraft before they occur. The FAA currently has a set of regulations governing minimum aircraft separation between aircraft flying under instrument flight rules. It would be useful to have a system that predicts the time when the loss in legal separation between aircraft occurs. The separation requirements may vary as a function of altitude of the aircraft. For instance, current FAA guidelines stipulate a horizontal separation of three nautical miles between aircraft on instruments below 10,000 feet (Figure 5.13). Above 10,000 feet, a separation of five miles is required. The required vertical separation between these aircraft is 1000 feet.

The separation volume for an aircraft can be modeled as a cylindrical volume centered about the aircraft, with a minimum lateral radius and a minimum altitude difference between aircraft. We can model this volume as a parametric cylinder moving as a function of time. Figure 5.14 shows an example set of five bounding volumes of aircraft within controlled airspace that must be monitored for potential loss of separation. In general, the aircraft may be ascending or descending, and may be moving at airspeeds ranging from 100 to 400 knots. Aircraft are restricted to less than 250 knots below 10,000 feet. The ϵ-collision system can determine potentially if and when a loss of separation will occur between aircraft, given the

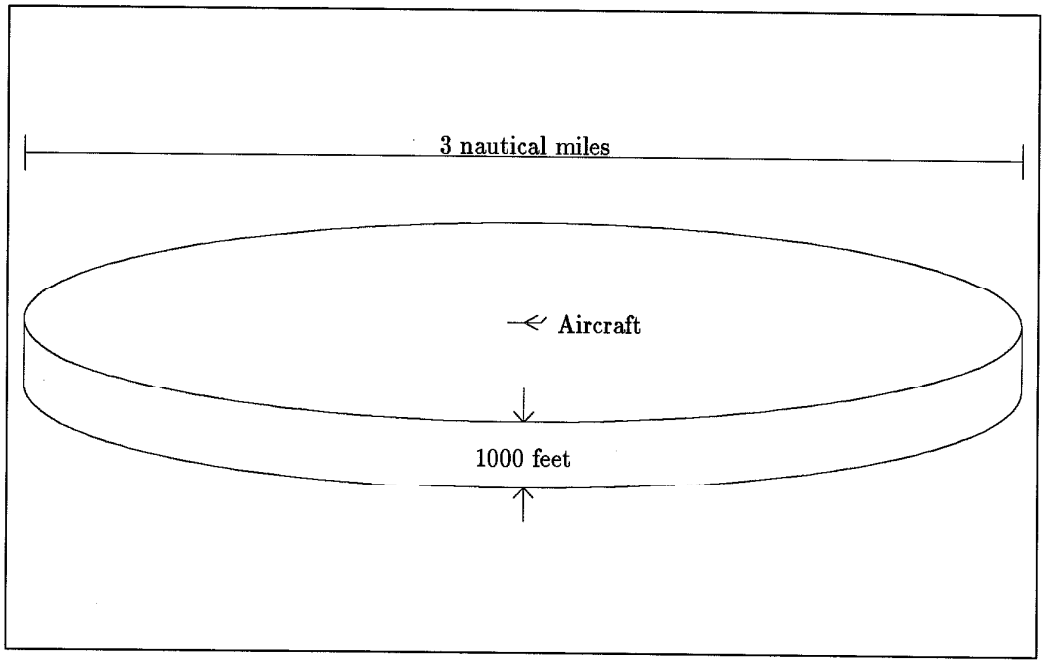

Figure 5.13: Scale illustration of the separation volume about a commercial airliner. We require that the separation volumes about aircraft do not collide, in order to maintain separation requirements mandated by the FAA. In general, the separation diameter of the volume is a function of the airspeed and altitude of the aircraft.

current trajectories and flight paths of the aircraft involved.

The rate constants for the aircraft collision problem are given by the maximum airspeed of each aircraft, and by the maximum rate at which the aircraft can ascend or descend. A parametric cylinder $\vec{C}(u,v,t)$ centered about the origin, of height h and radius r, may be defined as

$$\vec{C}(u,v,t) \equiv \begin{cases} \begin{pmatrix} 3vr\cos 2\pi u \\ 3vr\sin 2\pi u \\ -h/2 \end{pmatrix} & \text{if } v < 1/3 \\ \begin{pmatrix} r\cos 2\pi u \\ r\sin 2\pi u \\ 3h(v-1/2) \end{pmatrix} & \text{if } 1/3 \le v \le 2/3 \\ \begin{pmatrix} 3(1-v)r\cos 2\pi u \\ 3(1-v)r\sin 2\pi u \\ h/2 \end{pmatrix} & \text{if } v > 2/3. \end{cases} \tag{5.2}$$

The origin of this parametric cylinder is centered about the aircraft as it moves

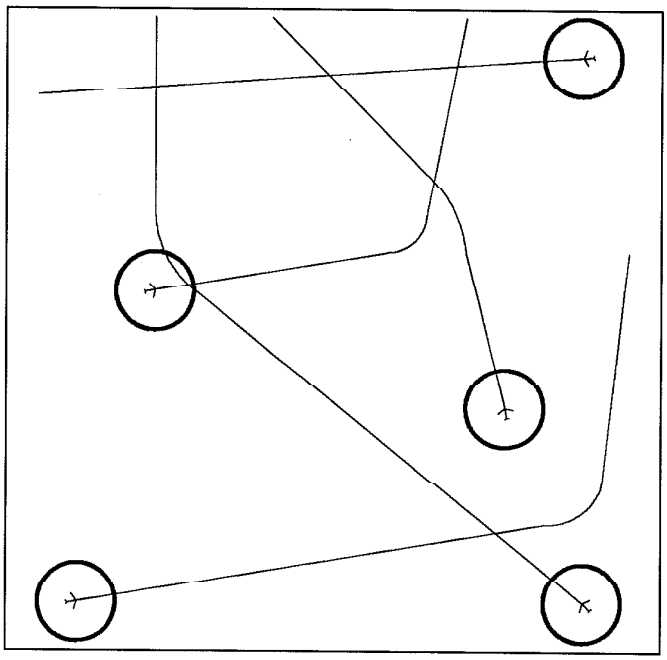

Figure 5.14: Overhead view of the aircraft separation problem. We need to preserve the separation between aircraft moving at a range of speeds and altitudes. A parametric cylinder serves as a good model for the airspace separation requirements mandated by the FAA. The paths of the aircraft are given as a function of time, and the predicted separation distance is determined by the parametric collision system.

along its planned trajectory.

In order to implement this system, we would compute each of the parametric derivatives for Eqn. 5.2. This would give us the Jacobian matrix for the function. Taking the maxima of the Jacobian over subregions would produce sufficient values for the rate matrix **M**. We would then apply the collision algorithm of Figure 4.7 to compute the separation distance between aircraft.

Appendix A

Theorems for ϵ-Collisions

Here we reiterate some of the commonly used definitions in the theorems.

Definition A.1 (ϵ-collision) *Given two parametric functions $\vec{f}(\vec{u},t)$, and $\vec{g}(\vec{u},t)$, and a distance tolerance ϵ, an ϵ-collision is defined by the condition*

$$\left\|\vec{f}(\vec{u}_f,t_0)-\vec{g}(\vec{u}_g,t_0)\right\|<\epsilon,$$

for some time t_0, some value of $\vec{u}_f$ in the domain of $\vec{f}(\vec{u},t)$, and some value of $\vec{u}_g$ in the domain of $\vec{g}(\vec{u},t)$.

Definition A.2 (rate matrix) *Given a continuous parametric, three-dimensional vector function $\vec{f}(u,v,t)$, and a rectangular region R about $(u_c,v_c,t_c)^T$ with half-widths Δu, Δv, and Δt:*

$$R=\left\{(u,v,t)^T:\ |u-u_c|\le\Delta u,\ \ |v-v_c|\le\Delta v,\ \ |t-t_c|\le\Delta t\ \right\}$$

in the domain of $\vec{f}(u,v,t)$; a rate matrix is a matrix $\mathbf{M}$ that satisfies for any two points $(u_1,v_1,t_1)^T$ and $(u_2,v_2,t_2)^T$ in region R:

$$\begin{aligned}
|x(u_1,v_1,t_1)-x(u_2,v_2,t_2)| &\le M_{xu}|u_1-u_2|+M_{xv}|v_1-v_2|+M_{xt}|t_1-t_2|,\\
|y(u_1,v_1,t_1)-y(u_2,v_2,t_2)| &\le M_{yu}|u_1-u_2|+M_{yv}|v_1-v_2|+M_{yt}|t_1-t_2|,\\
|z(u_1,v_1,t_1)-z(u_2,v_2,t_2)| &\le M_{zu}|u_1-u_2|+M_{zv}|v_1-v_2|+M_{zt}|t_1-t_2|,
\end{aligned}\tag{A.1}$$

where $(x,y,z)^T$ are the components of $\vec{f}$.

Definition A.3 (bounding box) *Given the conditions of Definition A.2, and a rate matrix M, we define a bounding box b_f of a function $\vec{f}(u,v,t)=(x,y,z)^T$*

over a region R to be

$$b_f = \left\{ \begin{pmatrix} u \\ v \\ t \\ x \\ y \\ z \end{pmatrix} : \begin{pmatrix} |u - u_{cf}| & \leq & \Delta u_f, \\ |v - v_{cf}| & \leq & \Delta v_f, \\ |t - t_{cf}| & \leq & \Delta t_f, \\ |x - x_{cf}| & \leq & \Delta x_f, \\ |y - y_{cf}| & \leq & \Delta y_f, \\ |z - z_{cf}| & \leq & \Delta z_f \end{pmatrix} \right\},$$

where

$$x_{cf} = \left(\vec{f}(u_c, v_c, t_c)\right)_x, \quad y_{cf} = \left(\vec{f}(u_c, v_c, t_c)\right)_y, \quad z_{cf} = \left(\vec{f}(u_c, v_c, t_c)\right)_z,$$

and

$$\begin{aligned} \Delta x_f &= M_{xu}\Delta u_f + M_{xv}\Delta v_f + M_{xt}\Delta t_f, \\ \Delta y_f &= M_{yu}\Delta u_f + M_{yv}\Delta v_f + M_{yt}\Delta t_f, \\ \Delta z_f &= M_{zu}\Delta u_f + M_{zv}\Delta v_f + M_{zt}\Delta t_f. \end{aligned}$$

The value x_{cf} is the x-coordinate of the center of box b_f, and Δx_f is the half-width of box b_f in the x-dimension. The values for the other dimensions have analogous interpretations.

A.1 Finding ϵ-Collisions

Theorem A.1 (ϵ-collision Theorem) *Given two time-dependent, continuous, three-dimensional vector functions $\vec{f}(u,v,t)$ and $\vec{g}(u,v,t)$, two parametric domains for these functions R_f, R_g, two rate matrices $\mathbf{M}_f, \mathbf{M}_g$ of $\vec{f}$ and $\vec{g}$, and a distance tolerance ϵ; and given two finite sets of bounding boxes, B_f and B_g, that completely contain surfaces $\vec{f}$ and $\vec{g}$, with the dimensions of each box $b_f \in B_f$ and $b_g \in B_g$ smaller than a given width δ; then it is possible to construct a set of ϵ-collisions between surfaces $\vec{f}$ and $\vec{g}$, and/or construct a set of $(\epsilon + 2\delta)$-collisions between surfaces $\vec{f}$ and $\vec{g}$, or else confirm that no ϵ-collision occurs. All ϵ-collisions are found, either as ϵ-collisions or as $(\epsilon + 2\delta)$-collisions.*[1]

Proof: We pair every box $b_f \in B_f$ with every box $b_g \in B_g$. If there is no time overlap between b_f and b_g, then there cannot be an ϵ-collision between those boxes.

[1]There is a region of uncertainty between confirming an ϵ-collision and confirming the lack of an ϵ-collision. We can make the region of uncertainty suitably small by choosing δ to be small enough. If we choose $\delta < \epsilon^2$, then the largest region of uncertainty lies between ϵ and $\epsilon + 2\epsilon^2$, which is a vanishingly small difference.

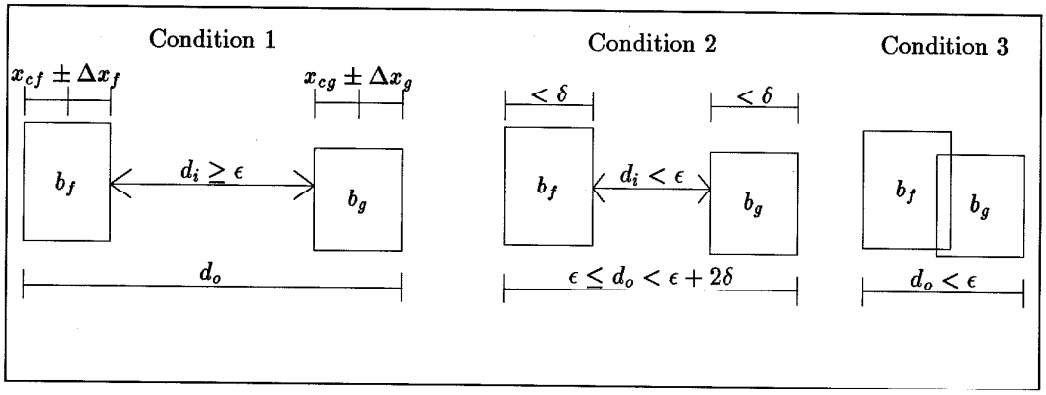

Figure A.1: Partitioning of the ϵ-collision theorem into three cases.

No time overlap exists if the earliest time of b_g is later than the latest time of b_f. This condition exists when

$$|t_{cf} - t_{cg}| > \Delta t_f + \Delta t_g.$$

In other words, when the central time difference of the two boxes exceeds the half-durations of the two boxes, there can be no time overlap.

We define an inner diameter d_i and an outer diameter d_o for a pair of boxes $\{b_f, b_g\}$ (see Figure A.1):

$$d_i \equiv \max \begin{pmatrix} |x_{cf} - x_{cg}| - \Delta x_f - \Delta x_g, \\ |y_{cf} - y_{cg}| - \Delta y_f - \Delta y_g, \\ |z_{cf} - z_{cg}| - \Delta z_f - \Delta z_g \end{pmatrix}, \tag{A.2}$$

$$d_o \equiv \max \begin{pmatrix} |x_{cf} - x_{cg}| + \Delta x_f + \Delta x_g, \\ |y_{cf} - y_{cg}| + \Delta y_f + \Delta y_g, \\ |z_{cf} - z_{cg}| + \Delta z_f + \Delta z_g \end{pmatrix}. \tag{A.3}$$

Note that $d_i \leq d_o$ for all pairs of boxes (Figure A.1), since every term of Eqn. A.2 is smaller than the corresponding term of Eqn. A.3.

Exactly one of three conditions holds for each pair $\{b_f, b_g\}$:

$$\begin{array}{llllll} 1.) & \epsilon & \leq & d_i & \leq & d_o \quad \text{(confirm no } \epsilon\text{-collision)} \\ 2.) & d_i & < & \epsilon & \leq & d_o \quad \text{(recurse)} \\ 3.) & d_i & \leq & d_o & < & \epsilon \quad \text{(confirm } \epsilon\text{-collision)} \end{array} \tag{A.4}$$

Figure A.1 shows the three possible cases graphically.

1. If the first condition holds, then every point in box b_f lies at least distance ϵ away from every point in box b_g, so there cannot be an ϵ-collision between b_f and b_g.

2. If the second condition holds, the inner diameter d_i is less than ϵ, and we also know that the maximum width of each box is less than δ. So the outer diameter d_o is less than $\epsilon + 2\delta$. If there is a time overlap, then we confirm an $(\epsilon + 2\delta)$-collision.

3. If the third condition holds, then every point in box b_f lies within distance ϵ from every point in box b_g. If there is a time overlap between b_f and b_g, then we confirm an ϵ-collision between b_f and b_g.

We evaluate the conditions in Eqn. A.4 for every unique pair of boxes $\{b_f, b_g\}$ from the two surfaces. If the first condition holds for all pairs, then we confirm that the two surfaces do not collide. Otherwise, we report the ϵ-collisions from the two surfaces. By sorting the ϵ-collisions according to $\max(t_{cf} - \Delta t_f, t_{cg} - \Delta t_g)$, we can find the earliest possible collision time and location of the two surfaces. □

A.2 Making Bounding Boxes

Theorem A.2 (Bounding Box Construction) *Given a time-dependent, continuous, three-dimensional vector function $\vec{f}(u, v, t)$, and given a maximum box half-width $\delta/2$; and given a parametric domain $R_f : |u - u_c| < \Delta u$, $|v - v_c| < \Delta v$, $|t - t_c| < \Delta t$, for $\vec{f}(u, v, t)$; and given a rate matrix $\mathbf{M}$ of $\vec{f}$; then it is possible to construct a finite set of bounding boxes B_f that completely contain the range of the function $\vec{f}$ over R_f, with the dimensions of each box $b_f \in B_f$ smaller than δ.*

Proof: We define the image $(x_c, y_c, z_c)^T$ of the central point $(u_c, v_c, t_c)^T$:

$$(x_c, y_c, z_c)^T = \vec{f}(u_c, v_c, t_c).$$

The coordinates $(x_c, y_c, z_c)^T$ form the center of a bounding box enclosing the function $\vec{f}$ over the entire region R_f. Let the half widths $(\Delta x, \Delta y, \Delta z)^T$ of the bounding box be given by the rate matrix $\mathbf{M}$:

$$\begin{aligned} \Delta x &\equiv M_{xu}\Delta u + M_{xv}\Delta v + M_{xt}\Delta t, \\ \Delta y &\equiv M_{yu}\Delta u + M_{yv}\Delta v + M_{yt}\Delta t, \\ \Delta z &\equiv M_{zu}\Delta u + M_{zv}\Delta v + M_{zt}\Delta t. \end{aligned} \tag{A.5}$$

If the maximum width of the bounding box is less than δ, i.e.

$$\max(\Delta x, \Delta y, \Delta z) < \delta/2,$$

then we have satisfied the conditions of the theorem.

Otherwise, we subdivide the parametric region R_f into two smaller parametric regions, using the k-d tree algorithm of Section 4.3.1. We choose the parametric axis for the split according to the largest value of D_u, D_v, or D_t:

$$\begin{aligned} D_u &\equiv (M_{xu} + M_{yu} + M_{zu})\Delta u \\ D_v &\equiv (M_{xv} + M_{yv} + M_{zv})\Delta v \\ D_t &\equiv (M_{xt} + M_{yt} + M_{zt})\Delta t. \end{aligned}$$

This determines the parametric dimension that is contributing most to the size of the bounding box in modeling space. This process is recursively invoked over smaller and smaller regions, until all the bounding boxes have widths less than δ.

We can guarantee that the recursion terminates, by examining the sum $\Delta x + \Delta y + \Delta z$. Since each term is non-negative, $\Delta x + \Delta y + \Delta z < \delta/2$ implies that each term separately is less than $\delta/2$. So it is sufficient to subdivide until the sum of the bounding box radii are less than $\delta/2$.

But we also have another formula for the sum of the bounding box radii:

$$\begin{aligned} \Delta x + \Delta y + \Delta z &= (M_{xu} + M_{yu} + M_{zu})\Delta u + \\ &\quad (M_{xv} + M_{yv} + M_{zv})\Delta v + \\ &\quad (M_{xt} + M_{yt} + M_{zt})\Delta t \\ &= D_u + D_v + D_t. \end{aligned}$$

Is is sufficient to consider the M_{ij} to be constant. The sum of the half-widths of the bounding box is now a function of three terms:

$$\Delta x + \Delta y + \Delta z = D_u + D_v + D_t.$$

The recursive procedure bisects the maximum of D_u, D_v, D_t, to produce a smaller bounding box. It is easy to show[2] that if we recurse three times using this method, then the sum $D_u + D_v + D_t$ must decrease by at least a factor of two, for non-negative D_u, D_v, D_t. This implies that for every three recursions, the sum $\Delta x + \Delta y + \Delta z$ decreases by at least a factor of two.

Starting with an initial box size of $\Delta x + \Delta y + \Delta z$, the recursive algorithm terminates when the box sizes are reduced to $\delta/2$. The ratio S of initial box size to final box size is given by

$$S \equiv \frac{\Delta x + \Delta y + \Delta z}{\delta/2}.$$

[2]If D_u, D_v, and D_t differ from each other by less than a factor of two, then each term is bisected once, and the sum is reduced by exactly a factor of two. If the terms differ by more than a factor of two, then the sum is reduced by more than a factor of two.

The recursion level N is bounded by

$$N \leq 3 \left\lceil \log_2 \left(\frac{\Delta x + \Delta y + \Delta z}{\delta/2} \right) \right\rceil .$$

This guarantees that the set of boxes of width smaller than δ will be finite; and the rate matrix $\mathbf{M}$ guarantees that each bounding box will bound its parametric subregion. Since we span all of parametric region R_f, the set of bounding boxes completely bounds the function $\vec{f}$ over R_f. □

A.3 Finding ϵ-Collisions Efficiently

We can combine the results of Appendix A.1 and Appendix A.2, to arrive at a more efficient approach to finding ϵ-collisions. As we construct the sets of bounding boxes through parametric subdivision, we can compare the boxes of surface $\vec{f}$ to the boxes of surface $\vec{g}$. If condition 1 of Eqn. A.4 is satisfied, then we know that the two parametric regions are disjoint and separated by at least distance ϵ. It is not necessary to subdivide these pairs of boxes further, since they do not collide. In this manner, it is possible to hierarchically examine the subregions of the two surfaces, and quickly determine if an ϵ-collision occurs. See Figure 4.7 for more details on the hierarchical algorithm.

A.4 Termination of the ϵ-Collision Algorithm

Lemma A.1 (ϵ-Collision Termination) *Given the conditions of Theorem A.1, ϵ-collision determination for two parametric surfaces terminates in a finite number of steps.*

Sketch of Proof: By the theorem in Appendix A.2, finite sets of bounding boxes can be constructed in a finite number of steps that completely bound L-functions. The procedure for determining ϵ-collisions does a comparison of all pairs of bounding boxes between the two surfaces. Since the sets of boxes are finite, so is the set of all unique pairs of bounding boxes. The algorithm goes through all the pairs and then terminates. Therefore, the algorithm terminates in a finite number of steps. □

A.5 Example of an Insoluble Collision Without the Lipschitz Condition

Lemma A.2 *Given only a finite number of parametric samples from a surface, with no additional information, it is not possible, in general, to solve the collision problem between arbitrary parametric surfaces.*

Sketch of Proof: Given object A, a unit sphere centered at the origin, and object B, a unit sphere centered at the point $\vec{P}$, we define object A to be motionless, and object B to be motionless, except at time t_1, when it moves to the origin. Then objects A and B collide at time t_1. With a finite set of samples across the time domain, it is impossible to find t_1, when t_1 is arbitrary. Therefore, it is impossible to determine if there is a collision, for arbitrary t_1. Therefore, the collision-determination problem is insoluble for arbitrary parametric functions.

Appendix B

Derivations for Adaptive Sampling

B.1 Derivation of Upper Bounds on Parametric Derivatives for a Sphere

We wish to find a sufficient function $L_u(R)$ that satisfies Eqn. 4.6 for a parametric sphere (Section 4.3.5). The parametric sphere is given by

$$\vec{f}(u,v) = \begin{pmatrix} \cos(2\pi u)\sin(\pi v) \\ \sin(2\pi u)\sin(\pi v) \\ -\cos(\pi v) \end{pmatrix}. \tag{B.1}$$

The condition on L_u is

$$L_u \geq \max_R \left\| \frac{\partial \vec{f}}{\partial u} \right\|_2 . \tag{B.2}$$

The partial derivatives with respect to u for $\vec{f}(u,v)$ are

$$\frac{\partial x}{\partial u} = -2\pi \sin(2\pi u)\sin(\pi v), \tag{B.3}$$

$$\frac{\partial y}{\partial u} = 2\pi \cos(2\pi u)\sin(\pi v), \tag{B.4}$$

$$\frac{\partial z}{\partial u} = 0. \tag{B.5}$$

We substitute into the equation for L_u:

$$L_u(R) = \max_R \left\| \frac{\partial \vec{f}}{\partial u} \right\|_2 \tag{B.6}$$

$$
\begin{aligned}
L_u(R) &= \max_R \left\| (2\pi \sin 2\pi u \sin \pi v, 2\pi \cos 2\pi u \sin \pi v, 0)^T \right\|_2 && \text{(B.7)} \\
&= \max_R \left\| 2\pi \sin(\pi v)\, (\sin 2\pi u, \cos 2\pi u, 0)^T \right\|_2 && \text{(B.8)} \\
&= \max_R |2\pi \sin \pi v| && \text{(B.9)} \\
&= \max_R |2\pi \cos \pi (v - 1/2)| && \text{(B.10)} \\
&= 2\pi \cos\left(\pi \min_R |v - 1/2|\right) && \text{(B.11)} \\
&= 2\pi \cos\left(\pi \max\left(0, |v_c - 1/2| - \Delta v\right)\right). \square && \text{(B.12)}
\end{aligned}
$$

B.2 Derivation of the Jacobian Matrix for a Parametric Sphere

Given a parametric sphere $\vec{f}(u, v, t)$,

$$
\vec{f}(u, v) = \begin{pmatrix} \cos(2\pi u) \sin(\pi v) \\ \sin(2\pi u) \sin(\pi v) \\ -\cos(\pi v) \end{pmatrix}, \tag{B.13}
$$

moving with non-constant velocity $\vec{s} = (s_x, s_y, s_z)$, find the rate matrix $\mathbf{M}$ defined to be

$$
\mathbf{M} \equiv \begin{pmatrix} \max_R |\partial x/\partial u| & \max_R |\partial x/\partial v| & \max_R |\partial x/\partial t| \\ \max_R |\partial y/\partial u| & \max_R |\partial y/\partial v| & \max_R |\partial y/\partial t| \\ \max_R |\partial z/\partial u| & \max_R |\partial z/\partial v| & \max_R |\partial z/\partial t| \end{pmatrix}. \tag{B.14}
$$

First, we compute the Jacobian of the function $\vec{f}(u, v, t)$, defined by

$$
\mathbf{J}(u, v, t) \equiv \begin{pmatrix} \dfrac{\partial x}{\partial u} & \dfrac{\partial x}{\partial v} & \dfrac{\partial x}{\partial t} \\ \dfrac{\partial y}{\partial u} & \dfrac{\partial y}{\partial v} & \dfrac{\partial y}{\partial t} \\ \dfrac{\partial z}{\partial u} & \dfrac{\partial z}{\partial v} & \dfrac{\partial z}{\partial t} \end{pmatrix}. \tag{B.15}
$$

Note that $\partial x/\partial t = s_x$, $\partial y/\partial t = s_y$, and $\partial z/\partial t = s_z$. Substituting,

$$
\mathbf{J}(u, v, t) = \begin{pmatrix} -2\pi \sin(2\pi u) \sin(\pi v) & \pi \cos(2\pi u) \cos(\pi v) & s_x(u, v, t) \\ 2\pi \cos(2\pi u) \sin(\pi v) & \pi \sin(2\pi u) \cos(\pi v) & s_y(u, v, t) \\ 0 & \pi \sin(\pi v) & s_z(u, v, t) \end{pmatrix}. \tag{B.16}
$$

The rate matrix $\mathbf{M}$ is just the maximum over region R of the absolute value of each component of the Jacobian:

$$\mathbf{M} \equiv \begin{pmatrix} \max_R |\partial x/\partial u| & \max_R |\partial x/\partial v| & \max_R |\partial x/\partial t| \\ \max_R |\partial y/\partial u| & \max_R |\partial y/\partial v| & \max_R |\partial y/\partial t| \\ \max_R |\partial z/\partial u| & \max_R |\partial z/\partial v| & \max_R |\partial z/\partial t| \end{pmatrix}. \tag{B.17}$$

Distributing the maxima and absolute value operations:

$$\mathbf{M}(R) = \begin{pmatrix} 2\pi \max_R |\sin 2\pi u| \max_R |\sin \pi v| & \pi \max_R |\cos 2\pi u| \max_R |\cos \pi v| & \max_R |s_x| \\ 2\pi \max_R |\cos 2\pi u| \max_R |\sin \pi v| & \pi \max_R |\sin 2\pi u| \max_R |\cos \pi v| & \max_R |s_y| \\ 0 & \pi \max_R |\sin \pi v| & \max_R |s_z| \end{pmatrix}.$$

B.3 Non-differentiable Surfaces with Lipschitz Constants

As an example, we examine the function diagramed in Figure B.1:

$$f(u) = \lim_{h \to 0} h \lfloor u/h \rfloor. \tag{B.18}$$

This function is not differentiable anywhere, yet it satisfies the Lipschitz condition:

$$|f(u_2) - f(u_1)| \leq L|u_2 - u_1| \tag{B.19}$$

for a value of $L = 1$.

B.4 Binary Right Triangles vs. Restricted Quadtrees

Figure B.2 illustrates the difference in efficiency between restricted quadtrees of squares vs. bintrees of right triangles. We subdivide the lower left corner of the parametric square in a recursive manner. For this example, the restricted quadtree requires exactly 3.5 times as many triangles as bintrees of right triangles, for the case of infinite recursion.

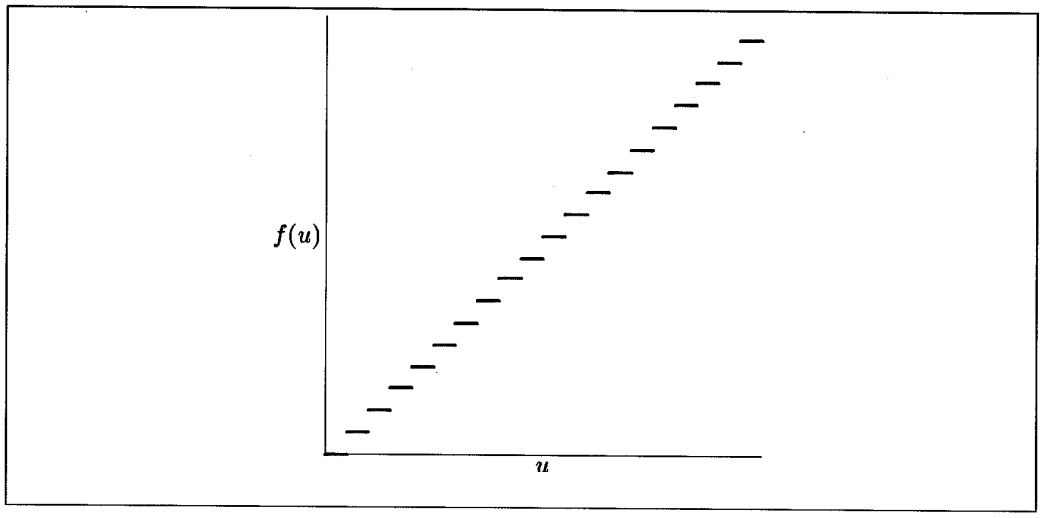

Figure B.1: Plot of $f(u) = \lim_{h \to 0} h\lfloor u/h \rfloor$, for a finite value of h.

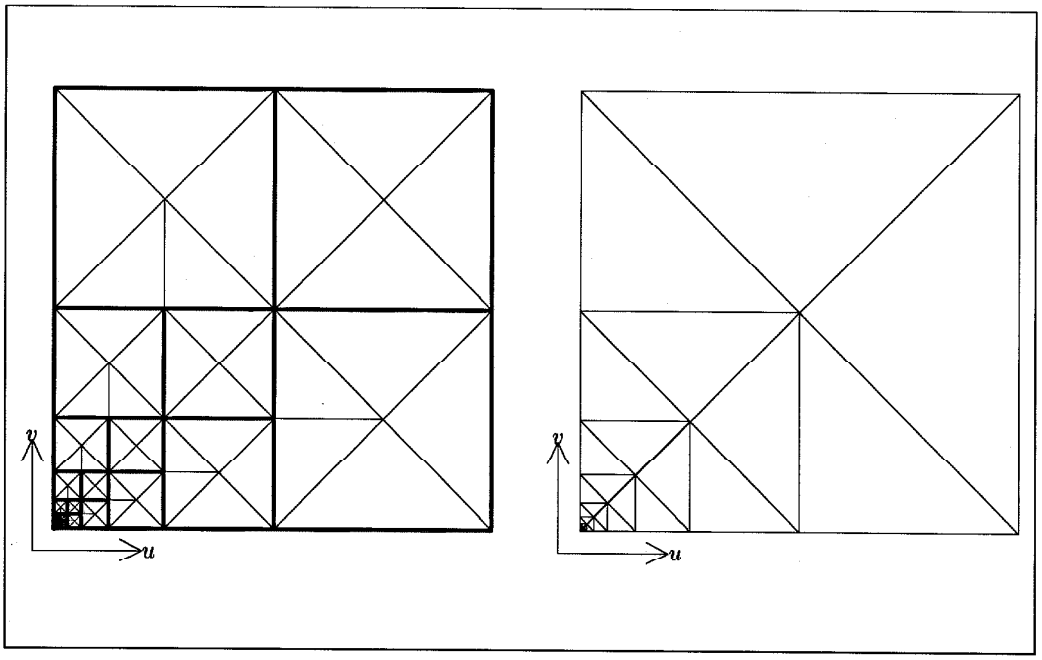

Figure B.2: Comparison of restricted quadtrees of squares (on the left) to bintrees of right triangles (on the right), for the case where we subdivide the lower left corners of the squares. Note that fewer corner samples are required for the bintree of right triangles. Yet we preserve the smooth variation in sampling frequency across the surface.

References

[Barr 83] Barr, Alan H., *Geometric Modeling and Fluid Dynamic Analysis of Swimming Spermatozoa,* Ph.D. Dissertation, Rensselaer Polytechnic Institute, 1983.

[Barr 84] Barr, Alan H., "Local and Global Deformations of Solid Primitives," *Computer Graphics 18*, 3, July 1984, 21–30.

[Barr 86] Barr, Alan H., "Ray Tracing Deformed Surfaces," *Computer Graphics 20*, 4, August 1986, 287–296.

[Barzel and Barr 88] Barzel, Ronen, and Alan H. Barr, "A Modeling System Based on Dynamic Constraints," *Computer Graphics 22,* 4, August 1988.

[Bentley and Friedman 79] Bentley, Jon L., and Jerome H. Friedman, "Data Structures for Range Searching," *ACM Computing Surveys 11,* 4, December 1979, 397–409.

[Besl and Jain 88] Besl, Paul J., and Ramesh C. Jain, "Segmentation through Variable-Order Surface Fitting," *IEEE Transactions on Pattern Analysis and Machine Intelligence 10,* 2, March 1988, 167–192.

[Bezier 74] Bezier, Pierre, "Mathematical and Practical Possibilities of UNISURF," in *Computer-Aided Geometric Design,* edited by Robert E. Barnhill and Richard F. Riesenfeld, Academic Press, New York, 1974, pp. 127–152.

[Blinn 78] Blinn, Jim, *Computer Display of Curved Surfaces,* Ph.D. Dissertation, University of Utah, 1978.

[Cameron and Culley 86] Cameron, S.A., and R.K. Culley, "Determining the Minimum Translational Distance Between Two Convex Polyhedra," *IEEE International Conference on Robotics and Automation,* 1986.

[Carlson 82] Carlson, Wayne E. "An Algorithm and Data Structure for 3D Object Synthesis using Surface Patch Intersections," *Computer Graphics 16,* 3, July 1982, 255.

[Catmull 75] Catmull, Ed, "Computer Display of Curved Surfaces," *IEEE Conference Proceedings on Computer Graphics, Pattern Recognition and Data Structures,* May 1975, 11.

[Crow 84] Crow, Franklin C., "Summed-Area Tables for Texture Mapping," *Computer Graphics 18,* 3, July 1984, 207–212.

[Culley and Kempf 86] Culley, R.K., and K.G. Kempf, "A Collision Detection Algorithm Based on Velocity and Distance Bounds," *Proceedings 1986 IEEE International Conference on Robotics and Automation,* Volume 2, pp. 1064–1069.

[Foley and Van Dam 82] Foley, James D., and Andries Van Dam, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley Publishing Co., Reading, Massachusetts, 1982, pp. 531–536.

[Gear 71] Gear, C. William, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971, p. 55.

[Gouraud 71] Gouraud, Henri, "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers 20,* 6, June 1971, 623–628.

[Kajiya and Snyder 88] Kajiya, Jim, and John Snyder, "Generative Modeling," *Computer Graphics 22,* 4, August 1988.

[Kay and Kajiya 86] Kay, Tim, and Jim Kajiya, "Ray Tracing Complex Scenes," *Computer Graphics 20,* 4, August 1986, 269.

[Kirkpatrick 83] Kirkpatrick, David, "Optimal Search in Planar Subdivisions," *SIAM Journal of Computing 12,* 1, February 1983, 28.

[Knuth 69] Knuth, Donald, *The Art of Computer Programming; Vol. 1, Fundamental Algorithms,* Addison-Wesley, Menlo Park, CA, 1969, Section 2.2.4.

[Lane and Carpenter 79] Lane, Jeff, and Loren Carpenter, "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing 11,* 1979, 290.

[Lane and Riesenfeld 80] Lane, Jeff, and Richard F. Riesenfeld, "A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence 2,* 1, January 1980, 35–46.

[Lee and Preparata 84] Lee, D. T. , and Franco P. Preparata, "Computational Geometry— A Survey," *IEEE Transactions on Computers C-33,* 12, December 1984, 1072.

[Lin and Segel 74] Lin C. C., and L. A. Segel, *Mathematics Applied to Deterministic Problems in the Natural Sciences,* Macmillan Publishing Co., Inc., New York, 1974, pp. 57–58.

[New York Times, June 19, 1988] Malcolm, Andrew H., "Aviation Experts Warn of Gridlock at U.S. Airports," *New York Times 137,* No. 47541, June 19, 1988, p. 1.

[Rao 84] Rao, S. S., *Optimization Theory and Applications,* Wiley Eastern Limited, New Delhi, India, 1984, pp. 248–249.

[Samet 82] Samet, Hanan "Neighbor Finding Techniques for Images Represented by Quadtrees," *Computer Graphics and Image Processing 18,* 1982, 37.

[Samet 84] Samet, Hanan "The Quadtree and Related Hierarchical Data Structures," *Computing Surveys 16,* 2, June 1984, 187–260.

[Schmitt, Barsky, Du 86] Schmitt, Francis, Brian Barsky, and Wen-Hui Du. "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data" *Computer Graphics 20,* 4, August 1986, 179–188.

[Schwarz 81] Schwarz, J.T., "Finding the Minimum Distance Between Two Convex Polygons," *Information Processing Letters 13,* 4, 1981, 168–170.

[Schweitzer and Cobb 82] Schweitzer, D., and E. S. Cobb, "Scanline Rendering of Parametric Surfaces," *Computer Graphics 16,* 3, July 1982, 265.

[Sederberg and Anderson 86] Sederberg, Tom, and David C. Anderson, "Ray Tracing of Steiner Patches," *Computer Graphics 18,* 3, July 1984, 159–164.

[Sederberg and Parry 86] Sederberg, Tom, and Scott Parry, "Free-Form Deformation of Solid Geometric Models," *Computer Graphics 20,* 4, August 1986, 151–160.

[Snyder and Barr 87] Snyder, John M., and Alan H. Barr, "Ray Tracing Complex Models Containing Surface Tessellations," *Computer Graphics 21,* 4, July 1987, 119–128.

[Swanson and Thayer 86] Swanson, Roger W., and Larry J. Thayer, "A Fast Shaded-Polygon Renderer," *Computer Graphics 20,* 4, August 1986, 95–102.

[Von Herzen 85] Von Herzen, Brian, *Sampling Deformed, Intersecting Surfaces with Quadtrees,* Masters Thesis, California Institute of Technology, Computer Science Dept., 5179:TR:85, 1985.

[Warnock 69] Warnock, J.E., *A Hidden-Line Algorithm for Halftone Picture Representation,* Ph.D. Dissertation, University of Utah, 1969.

[Whitted 80] Whitted, Turner, "An Improved Illumination Model for Shaded Display," *Communications ACM 23,* 6, June 1980, 343.

[Williams 83] Williams, Lance, "Pyramidal Parametrics," *Computer Graphics 17,* 3, July 1983, 1–11.